
Car Connectivity Consortium

Digital Key Technical Specification

Version 4.0.0
(CCC-TS-101)



Copyright © 2011-2025 Car Connectivity Consortium LLC
All rights reserved
Confidential

1 VERSION HISTORY

Version	Date	Comment
1.0	16 Apr 2020	Release 2, BoD Approved Version 1.0
1.1	22 Dec 2020	Release 2, BoD Approved Version 1.1
1.0	19 May 2021	Release 3, BoD Approved Version 1.0
1.2.2	Jul 7 th 2023	Version approved by TWG Version 1.2.1-r3 renamed to version 1.2.2
1.2.3	Nov 6 th 2023	Version approved by TWG Version 1.2.2-r2 renamed to version 1.2.3
4.0.0	Mar 19, 2025	Release 4.0.0, BoD Approved Version 4.0.0

LEGAL NOTICE

The copyright in this Specification is owned by the Car Connectivity Consortium LLC (“CCC LLC”). Use of this Specification and any related intellectual property (collectively, the “Specification”) is governed by; (a) the license terms contained in this Legal Notice and the Car Connectivity Consortium Specification License Agreement (the “License Agreement”) for anyone who is not a Member of CCC LLC (a “non-Member”); or (b) the terms contained in this Legal Notice and the CCC LLC Limited Liability Company Agreement (the “LLC Agreement”) for any party who is a Member of CCC LLC (a “Member”).

Use of the Specification by a non-Member is prohibited unless such non-Member has entered into the License Agreement and downloaded this Specification in accordance with the requirements of CCC LLC. The legal rights and obligations of each Member are governed by the LLC Agreement and their applicable Membership Agreement, including without limitation those contained in Article 10 of the LLC Agreement.

In addition to the terms of the LLC Agreement and its exhibits and appendices, CCC LLC hereby grants each Member a right to use and to make verbatim copies of the Specification for the purposes of implementing the technologies specified in the Specification to their products (“Implementing Products”) under the terms of the LLC Agreement (the “Purpose”). Members are not permitted to make available or distribute this Specification or any copies thereof to non-Members other than to their Affiliates (as defined in the LLC Agreement) and subcontractors but only to the extent that such Affiliates and subcontractors have a need to know for carrying out the Purpose and provided that such Affiliates and subcontractors accept confidentiality obligations similar to those contained in the LLC Agreement. Each Member shall be responsible for the observance and proper performance by such of its Affiliates and subcontractors of the terms and conditions of this Legal Notice and the Agreement. No other license, express or implied, by estoppel or otherwise, to any intellectual property rights are granted to Members herein.

THE SPECIFICATION IS PROVIDED “AS IS” WITH NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS, AND COMPLIANCE WITH APPLICABLE LAWS. Notice for Members: Any use of the Specification not in compliance with the applicable terms of this Legal Notice, the LLC Agreement, and Membership Agreement is prohibited and any such prohibited use may result in termination of the applicable Membership Agreement and other liability permitted by the applicable agreement or by applicable law to CCC LLC or any of its members for patent, copyright, and/or trademark infringement.

Notice for Non-Members: Any use of the Specification not in compliance with the applicable terms of this Legal Notice or the License Agreement for non-Members is prohibited and any such prohibited use may result in termination of the non-Member’s License Agreement and other liability permitted by the License Agreement or by applicable law to CCC LLC or any of its members for patent, copyright, and/or trademark infringement.

NOTHING IN THE SPECIFICATION CREATES ANY WARRANTIES, EITHER EXPRESS OR IMPLIED, REGARDING SUCH LAWS OR REGULATIONS. ALL LIABILITY, INCLUDING LIABILITY FOR INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHTS OR FOR NONCOMPLIANCE WITH LAWS RELATING TO USE OF THE SPECIFICATION IS EXPRESSLY DISCLAIMED. BY USE OF THE SPECIFICATION, EACH MEMBER AND NON-MEMBER EXPRESSLY WAIVES ANY CLAIM AGAINST CCC LLC AND ITS MEMBERS RELATED TO USE OF THE SPECIFICATION.

Disclaimers for Members: Each Member hereby acknowledges that its Implementing Products may be subject to various regulatory controls under the laws and regulations of various jurisdictions worldwide. Such laws and regulatory controls may govern, among other things, the combination, operation, use, implementation, and distribution of Implementing Products. Examples of such laws and regulatory

controls include, but are not limited to, road safety regulations, telecommunications regulations, technology transfer controls, and health and safety regulations. Each Member is solely responsible for the compliance by their Implementing Products with any such laws and regulations and for obtaining any and all required authorizations, permits, or licenses for their Implementing Products related to such regulations within the applicable jurisdictions. Each Member acknowledges that nothing in the Specification provides any information or assistance in connection with securing such compliance, authorizations, or licenses.

Disclaimers for non-Members: Each non-Member that downloads or otherwise obtains access to the Specification acknowledges that its limited copyright license is for the sole purpose of evaluation and such license does not extend to implementations. Products that are the result of implementing the Specification may be subject to various regulatory controls under the laws and regulations of various jurisdictions worldwide. Such laws and regulatory controls may govern, among other things, the combination, operation, use, implementation, and distribution of products. Examples of such laws and regulatory controls include, but are not limited to, road safety regulations, telecommunications regulations, technology transfer controls, and health and safety regulations. Each non-Member that downloads or otherwise obtains access to the Specification is solely responsible for decisions whether to become a Member of CCC LLC and whether to implement products based upon the Specification, and non-Member acknowledges that if such non-Member implements products then it is solely responsible for the compliance with any such laws and regulations and for obtaining any and all required authorizations, permits, or licenses for their products related to such regulations within the applicable jurisdictions. Each non-Member that downloads or otherwise obtains access to the Specification acknowledges that nothing in the Specification provides any information or assistance in connection with securing such compliance, authorizations, or licenses for products that may implement the Specification.

CCC LLC reserves the right to adopt any changes or alterations to the Specification as it deems necessary or appropriate.

COPYRIGHT © 2011-2025. CCC LLC.

1 **TABLE OF CONTENTS**

2	VERSION HISTORY.....	2
3	LEGAL NOTICE	3
4	TABLE OF CONTENTS	5
5	LIST OF FIGURES.....	18
6	LIST OF TABLES.....	21
7	LIST OF LISTINGS.....	29
8	ABBREVIATIONS AND ACRONYMS.....	32
9	DEFINITIONS.....	35
10	NOTATIONS	39
11	CODE LISTINGS.....	41
12	1 INTRODUCTION AND SCOPE.....	42
13	1.1 RELEASE 4 FEATURE SUMMARY	42
14	1.1.1 Fleet Key Sharing	43
15	1.1.2 Delegated Key Sharing.....	43
16	1.1.3 Key Classes	43
17	1.1.4 Owner Sharing Control.....	43
18	1.1.5 Summary.....	43
19	2 SYSTEM ARCHITECTURE.....	45
20	2.1 OVERVIEW	45
21	2.2 HIGH-LEVEL FEATURES.....	45
22	2.3 HIGH LEVEL ARCHITECTURE	45
23	2.4 ACTORS	53
24	2.4.1 Vehicle	53
25	2.4.2 Vehicle NFC Readers [WCC1]	54
26	2.4.3 Vehicle Bluetooth LE (BLE) Module [WCC2/WCC3]	54
27	2.4.4 Vehicle UWB Module [WCC3]	54
28	2.4.5 Vehicle OEM Server	54
29	2.4.6 Key Tracking Server (KTS)	55
30	2.4.7 Devices	55
31	2.4.8 Device OEM Server	56
32	2.4.9 Relay Server	56
33	2.4.10 Server Based Owner Device (SBOD).....	56
34	2.4.11 Server Based Friend Device (SBFD)	56
35	2.4.12 Fleet Management Server (FMS)	57
36	2.4.13 Fleet Management App	57
37	2.4.14 Key Issuing Server (KIS).....	57
38	2.4.15 Service Provider Server (SPS).....	57
39	2.4.16 Service Provider App.....	58

1	2.5	RELATIONSHIPS	58
2	2.5.1	Door NFC Reader (3) [WCC1]	58
3	2.5.2	Console NFC Reader (4) [WCC1]	59
4	2.5.3	Bluetooth LE Interface (11) [WCC2/WCC3]	59
5	2.5.4	UWB Interface (12) [WCC3]	59
6	2.5.5	Sender-to-Receiver Device Link (via (2), (6), (8), and (7))	59
7	2.5.6	Sender or Receiver Device to Vehicle OEM Server (10, 9)	59
8	2.5.7	Telematics Link (1)	60
9	2.5.8	Sender/Receiver Device OEM Server Link (2,7)	60
10	2.5.9	Vehicle OEM Server to KTS (5)	60
11	2.5.10	Sender/Receiver Device OEM Server to Vehicle OEM Server (6,8)	60
12	2.5.11	Fleet Management Server – Server-Based Owner Device Link (21)	60
13	2.5.12	Vehicle OEM Server – Server-Based Owner Device Link (20)	61
14	2.5.13	Receiver Device OEM Server – Vehicle OEM Server Link (8) – Server-based	
15		Owner Device Server Link (20)	61
16	2.5.14	Server-Based Owner Device – Relay Server Link (16) – Receiver Device OEM	
17		Server Link (14)	61
18	2.5.15	Receiver Device – Relay Server Link (14) – Server-Based Friend Device Link (19)	
19		61	
20	2.5.16	Server-Based Friend Device – Vehicle OEM Server Link (17)	61
21	2.5.17	Receiver Device OEM Server – Vehicle OEM Server Link (8)	61
22	2.6	DEVICE STRUCTURE	62
23	2.6.1	NFC Component [WCC1]	62
24	2.6.2	BLE Module [WCC2/WCC3]	63
25	2.6.3	UWB Module [WCC3]	63
26	2.6.4	Secure Element (or equivalent)	63
27	2.6.5	Digital Key Applet	63
28	2.6.6	Digital Key Framework	64
29	2.6.7	Vehicle OEM App	64
30	2.6.8	Native App	64
31	2.7	VEHICLE STATES	64
32	2.8	DIGITAL KEY USER ROLES	65
33	2.8.1	Owner Role	65
34	2.8.2	Shared Digital Key Roles	65
35	2.8.3	General Rules for all Digital Key Roles	65
36	2.8.4	Account Role Definition	66
37	2.9	ACCESS PROFILES	70
38	2.10	VERSIONING	71
39	2.10.1	General	71
40	2.10.2	Domain Versions	71
41	2.10.3	Domain Wise Version Agreement	73
42	2.10.4	Software Update Scenarios	76

1	2.10.5	Migration Scenarios	80
2	2.10.6	Version Numbers.....	88
3	2.10.7	Version Introduction.....	88
4	2.10.8	Version Support	89
5	2.10.9	Version Table	90
6	2.10.10	Version Deprecation.....	92
7	2.10.11	Digital Key Conversion	93
8	3	NFC INTERFACE [WCC1].....	95
9	3.1	NFC FUNCTIONAL REQUIREMENTS	95
10	3.1.1	Vehicle	95
11	3.1.2	Device	95
12	3.2	NFC PROCEDURES	96
13	3.2.1	NFC Polling and Link Setup Procedure	96
14	3.2.2	NFC Data Transfer Procedure	97
15	3.2.3	NFC Link Teardown Procedure	97
16	3.2.4	NFC Reset Procedure.....	98
17	4	DATA STRUCTURE.....	99
18	4.1	APPLET INSTANCE LAYOUT	99
19	4.2	DIGITAL KEY STRUCTURE	100
20	4.3	MAILBOXES	102
21	4.3.1	Private Mailbox	103
22	4.3.2	Confidential Mailbox.....	110
23	4.3.3	Immobilizer Token	111
24	4.3.4	Mailbox Access Rights.....	111
25	5	OWNER PAIRING COMMANDS [WCC1/WCC2/WCC3].....	112
26	5.1	SUPPORTED COMMANDS FOR OWNER PAIRING	112
27	5.1.1	SELECT Command.....	113
28	5.1.2	SPAKE2+ REQUEST Command.....	114
29	5.1.3	SPAKE2+ VERIFY Command	116
30	5.1.4	WRITE DATA Command.....	118
31	5.1.5	GET DATA Command	126
32	5.1.6	GET RESPONSE Command	127
33	5.1.7	OP CONTROL FLOW Command.....	128
34	5.2	DATA ELEMENTS.....	130
35	5.2.1	Certificates	130
36	6	OWNER PAIRING.....	131
37	6.1	OVERVIEW	131
38	6.2	KEYS AND DATA	131
39	6.3	OWNER PAIRING IMPLEMENTATION	132
40	6.3.1	Phase 0: Preparation	133
41	6.3.2	Phase 1: Initiate Pairing Procedure	134

1	6.3.3	Phase 2: First Session with NFC Reader.....	135
2	6.3.4	Phase 3: Second Session with NFC Reader.....	147
3	6.3.5	Phase 4: Finalization of Pairing Procedure.....	153
4	6.3.6	Timers.....	156
5	6.3.7	Pairing Password URL	157
6	7	STANDARD TRANSACTION.....	159
7	8	FAST TRANSACTION.....	161
8	9	USER AUTHENTICATION.....	162
9	9.1	EXPLICIT USER AUTHENTICATION POLICY	163
10	9.2	IMPLICIT USER AUTHENTICATION POLICY.....	164
11	10	CHECK PRESENCE TRANSACTION	165
12	11	DIGITAL KEY SHARING [WCC1/WCC2].....	166
13	11.1	ENCODING	166
14	11.2	SHARING PRINCIPLES	166
15	11.2.1	Definitions	166
16	11.2.2	Primary Sharing Principles	168
17	11.2.3	Sharing in a Chain Principles.....	169
18	11.2.4	Server-Based Key Sharing Considerations.....	170
19	11.3	COMMUNICATION CHANNEL	170
20	11.3.1	Notifications	171
21	11.3.2	Cross-Platform Sharing Invitation	171
22	11.3.3	General API Parameter Definitions	172
23	11.3.4	API Parameter Usage.....	176
24	11.3.5	Security Considerations	177
25	11.4	INTER-ACCOUNT SHARING KEY SHARING FLOW: STEPS	179
26	11.4.1	Steps 1 through 5 (Sender): Sharing Invitation	179
27	11.4.2	Steps 6 through 10 (Receiver): Key Signing Request.....	186
28	11.4.3	Steps 11 through 13 (Sender): Generate Import Request	188
29	11.4.4	Steps 14 and 15 (Receiver): Endpoint Data Import	192
30	11.4.5	Step 18 (Receiver): Track Key	194
31	11.4.6	Step 19 (Vehicle OEM Server): Track Key Response	194
32	11.4.7	(Shared Key): First Transaction.....	195
33	11.5	ONLINE SHARING PIN FLOW.....	196
34	11.5.1	Concept.....	196
35	11.5.2	Online Sharing PIN Flow: Steps 1 through 8:.....	198
36	11.5.3	Online Sharing PIN Flow: Steps 9 through 13:	198
37	11.5.4	Online Sharing PIN Flow: Steps 14 through 27: PIN validation.....	199
38	11.5.5	Online Sharing PIN Flow: Steps 28 to 36: Key Sharing finalization.....	199
39	11.6	DEVICE PIN.....	199
40	11.6.1	Concept.....	199
41	11.7	KEY TRACKING AND ONLINE ATTESTATION DELIVERY.....	202

1	11.7.1	Online immobilizer token and slot identifier retrieval	205
2	11.7.2	Online BLE Keys retrieval	205
3	11.8	OWNER DEVICE OEM SERVER NOTIFICATION	206
4	11.8.1	Vehicle OEM Server: eventNotification	206
5	11.8.2	Owner Device OEM Server: eventNotification() Response	206
6	11.9	KEY CONFIGURATION	207
7	11.10	INTRA-ACCOUNT SHARING WITH SECURE RECEIVER VERIFICATION	208
8	11.10.1	Steps 1 to 6: Initiate Sharing.....	208
9	11.10.2	Step 7 and 8: Key Sharing.....	208
10	11.10.3	Steps 9 to 12: Account Info Hash verification	210
11	11.11	CERTIFICATE CHAIN	210
12	11.12	KEY CLASSES	211
13	11.13	PREREQUISITES.....	215
14	11.14	ERROR CODES	216
15	11.15	ALGORITHMS.....	217
16	11.16	VERSIONING	218
17	12	SERVER-BASED KEY SHARING.....	220
18	12.1	SERVER PREPARATION	220
19	12.1.1	FMS/SBOD Connection Preparation (example)	220
20	12.1.2	FMS/SBOD/Vehicle OEM Server Preparation (example).....	220
21	12.2	VEHICLE INFLEETING	221
22	12.2.1	Proof of Ownership	221
23	12.2.2	Infleeting Flow	221
24	12.3	DIGITAL KEY SHARING FOR FLEET VEHICLES	223
25	12.3.1	Receiver device Binding.....	223
26	12.3.2	Receiver device Sharing App Access Rights	224
27	12.3.3	Digital Key Sharing Flow	224
28	12.4	DELEGATED KEY SHARING.....	225
29	12.4.1	SBFD Enablement	225
30	12.4.2	SBFD service activation.....	226
31	12.4.3	Service Key issuance from SBFD to Service User.....	228
32	12.4.4	Deletion of keys shared by SBFD	229
33	12.4.5	Service Cancellation/SBFD Key Deletion:	230
34	12.4.6	Owner Sharing Control	231
35	13	KEY TERMINATION AND DELETION [WCC1/WCC2].....	236
36	13.1	KEY TERMINATION SCENARIOS.....	237
37	13.2	SHARED KEY REMOTE TERMINATION	241
38	13.2.1	REV_100: Shared key termination in vehicle	241
39	13.2.2	REV_110: Shared key termination in owner Vehicle OEM account.....	243
40	13.2.3	REV_120: Shared key termination in receiver Vehicle OEM account	243

1	13.2.4	REV_130: Shared key termination on owner device natively.....	245
2	13.2.5	REV_140: Shared key termination on owner device in Vehicle OEM app.....	245
3	13.2.6	REV_150: Shared key termination based on expiry date of the key.....	246
4	13.2.7	REV_160: Shared key termination by Device OEM (device security issue).....	247
5	13.2.8	REV_170: Shared key termination due to remote wipe of device	249
6	13.2.9	REV_180: Shared key termination due to deletion of personal data in receiver Vehicle OEM account.....	251
8	13.3	SHARED KEY LOCAL TERMINATION	251
9	13.3.1	REV_200: Shared key termination on receiver device natively	251
10	13.3.2	REV_210: Shared key termination on receiver device in Vehicle OEM app.....	251
11	13.3.3	REV_220: Shared key termination due to local wipe of device.....	252
12	13.4	SHARED KEY REMOTE SUSPEND/RESUME	253
13	13.4.1	REV_300: Shared key temporary suspension in Device OEM account.....	253
14	13.4.2	REV_310: Shared key resume in Device OEM account.....	254
15	13.4.3	Resume attestation	256
16	13.5	OWNER KEY REMOTE TERMINATION	257
17	13.5.1	REV_400: Change owner device	257
18	13.5.2	REV_410: Owner key termination by Device OEM (security issue).....	259
19	13.5.3	REV_420: Owner key termination due to remote wipe of device	259
20	13.5.4	REV_500: Owner key termination on owner device natively (delete pass for Digital Key).....	259
22	13.5.5	REV_510: Owner key termination on owner device in Vehicle OEM app	259
23	13.5.6	REV_520: Owner key termination due to local wipe of device	259
24	13.5.7	REV_600/610: Owner key temporary suspension in device lost mode in Device OEM account.....	259
26	13.6	OWNER DEVICE UNPAIRING	261
27	13.6.1	REV_700: Unpairing in vehicle UI (sell vehicle).....	261
28	13.6.2	REV_710: Unpairing on owner device in Vehicle OEM app	263
29	13.6.3	REV_720: Unpairing in owner Vehicle OEM account (sale of vehicle).....	264
30	13.6.4	REV_730: Unpairing based on cancellation or expiry date of the Digital Key service	264
32	13.6.5	REV_740: Unpairing by Vehicle OEM (vehicle stolen).....	264
33	13.6.6	REV_750: Unpairing by Vehicle OEM (security breach in vehicle).....	264
34	13.6.7	REV_760: Unpairing by Vehicle OEM (garage service process)	265
35	13.6.8	REV_770: Unpairing due to deletion of personal data in owner Vehicle OEM account	265
37	13.6.9	REV_800: Unbinding of vehicle from owner account in vehicle UI (unpairing, owner sale of vehicle).....	265
39	13.6.10	REV_810: Unbinding of vehicle from owner account in owner Vehicle OEM account (unpairing, owner sale of vehicle).....	265
41	13.6.11	REV_820: Unbinding of vehicle from owner account in Vehicle OEM app (unpairing, owner sale of vehicle).....	265
43	13.7	TERMINATION IN VEHICLE.....	265

1	13.7.1	Rules	265
2	13.8	TERMINATION AND DELETION OF FMS USE CASES	266
3	13.8.1	Key Termination Scenarios	266
4	13.8.2	Use Case/Flow Mapping	267
5	13.9	EVENT NOTIFICATION: SBOD FORWARDING	267
6	13.10	DIGITAL KEY SUSPENSION/TERMINATION ON DEVICE	268
7	13.10.1	REV_FMS_100: Vehicle user terminates Receiver DK in vehicle OEM app	268
8		This flow corresponds to REV_210 with vehicle user as “Receiver”.	268
9	13.10.2	REV_FMS_110: Vehicle user terminates Shared Digital Key in native app	268
10		This flow corresponds to REV_200 with vehicle user as “Receiver”.	268
11	13.10.3	REV_FMS_120: FMS terminates Shared Digital Key vehicle user	268
12	13.10.4	REV_FMS_130: Shared key termination based on expiry date of the Digital Key	
13		268	
14	13.10.5	REV_FMS_140: Shared key termination due to local wipe of device	268
15	13.10.6	REV_FMS_170: Receiver reports lost/stolen receiver device: remote wipe keys	
16		268	
17	13.10.7	REV_FMS_180: Security breach on receiver device	269
18	13.10.8	REV_FMS_190: Vehicle user terminates Digital Key in the vehicle	269
19	13.10.9	REV_FMS_150: Receiver reports lost/stolen receiver device: suspend keys	269
20	13.10.10	REV_FMS_160: Shared key resume from device lost mode in device OEM	
21		account 269	
22	13.11	VEHICLE UNPAIRING	269
23	13.11.1	REV_FMS_500: Unpairing based on cancellation or expiry date of the digital key	
24		service to FMS	269
25	13.11.2	REV_FMS_501: Unpairing by vehicle OEM (vehicle stolen, garage service,...)	
26		269	
27	13.11.3	REV_FMS_502: Unpairing by vehicle OEM (FMS security breach)	269
28	13.11.4	REV_FMS_504: FMS requests vehicle unpairing (vehicle de-fleeting)	269
29	14	AUTHENTICATION AND PRIVACY	272
30	14.1	AUTHENTICATION AND PRIVACY KEYS	272
31	14.1.1	Usage	272
32	14.1.2	Certificate Chains	273
33	14.2	REMOTE TERMINATION REQUESTS	275
34	14.2.1	Example of RTR Signature Verification	279
35	14.3	ENCRYPTION AND SIGNATURE VERIFICATION SCHEMES	280
36	14.4	OEM APP DATA ATTESTATION	280
37	15	DIGITAL KEY APPLET	281
38	15.1	INTRODUCTION	281
39	15.2	KEYS AND DATA	281
40	15.3	APPLET IMPLEMENTATION	282
41	15.3.1	Introduction	282

1	15.3.2	Commands.....	294
2	15.3.3	Security.....	340
3	15.3.4	Optimization.....	343
4	15.4	HELPER METHOD	343
5	15.4.1	Hierarchy	344
6	15.5	VEHICLE IMPLEMENTATION.....	354
7	15.5.1	Security Guidelines.....	354
8	15.5.2	Optimizations	355
9	16	CERTIFICATE.....	356
10	16.1	GENERAL	356
11	16.2	CERTIFICATES AND RELATIONSHIPS.....	356
12	16.2.1	[A] – SE Root CA Certificate	359
13	16.2.2	[B] – SE Root Certificate	359
14	16.2.3	[C] – Instance CA Attestation (signed by SE Root) per Vehicle OEM	359
15	16.2.4	[D] – Device OEM CA Certificate	359
16	16.2.5	[E] – Instance CA Certificate (Signed by Device OEM) per Vehicle OEM	360
17	16.2.6	[F] – Device OEM CA Certificate (Signed by Vehicle OEM)	360
18	16.2.7	[G] – Digital Key per vehicle	360
19	16.2.8	[H] – Digital Key Certificate	360
20	16.2.9	[J] – Vehicle OEM CA Certificate.....	360
21	16.2.10	[K] – Vehicle Public Key Certificate.....	361
22	16.2.11	[L] – Digital Key Creation Data	361
23	16.2.12	[M] – Vehicle OEM CA Certificate (signed by Device OEM).....	361
24	16.2.13	[N] – Instance CA Attestation (self-signed) per Vehicle OEM.....	361
25	16.2.14	Server Certificate Relationships	362
26	16.3	CERTIFICATE SIZE RESTRICTIONS	363
27	16.4	DEVICE CERTIFICATE CHAIN	364
28	16.5	VEHICLE CERTIFICATE CHAIN	365
29	16.6	SUPPORTED VERIFICATION CHAIN	365
30	16.7	OWNER PAIRING CERTIFICATE CHAIN	365
31	16.7.1	Device and Vehicle	365
32	16.8	KEY SHARING CERTIFICATE CHAIN	365
33	17	SERVER-TO-SERVER COMMUNICATIONS AND API.....	367
34	17.1	INTRODUCTION.....	367
35	17.2	API DESIGN.....	367
36	17.3	SECURITY	368
37	17.4	VERSIONS	368
38	17.5	URL SCHEME	368
39	17.6	USER INTERFACE.....	368
40	17.6.1	Description.....	368

1	17.7	APIs IMPLEMENTED BY VEHICLE OEM SERVER	369
2	17.7.1	API – migrationMetadata.....	369
3	17.7.2	API - recoverKeyData	371
4	17.7.3	API - trackKey.....	374
5	17.7.4	API – inFleet	377
6	17.7.5	API – registerKey	379
7	17.7.6	API – preShare.....	379
8	17.7.7	API – preTrack.....	381
9	17.8	APIs IMPLEMENTED BY DEVICE OEM SERVER AND VEHICLE OEM SERVER	383
10	17.8.1	API - healthCheck.....	383
11	17.8.2	API - manageKey.....	383
12	17.8.3	API – versionUpdate	385
13	17.9	APIs IMPLEMENTED BY DEVICE OEM SERVER	388
14	17.9.1	API – eventNotification	388
15	17.10	APIs IMPLEMENTED BY SERVER BASED DEVICES	391
16	17.10.1	API – infleetExternal	391
17	17.10.2	API – prepareKeySharingExternal.....	392
18	17.10.3	API – manageKeyExternal.....	393
19	17.10.4	API – eventNotificationExternal	395
20	17.10.5	API – getKeyInformation	395
21	17.10.6	API - defleetExternal	396
22	17.10.7	API – cancelService.....	397
23	17.11	DATA STRUCTURES.....	397
24	17.11.1	Objects.....	397
25	17.11.2	Enums	410
26	17.11.3	Server Status Codes	413
27	17.12	ECC ENCRYPTION FOR DEVICE AND SERVER	417
28	17.12.1	Frame/Package Contents	417
29	17.12.2	Encryption Process.....	417
30	17.12.3	Decryption Process.....	417
31	17.12.4	Key Derivation Function.....	418
32	17.12.5	Symmetric Encryption Key.....	419
33	17.12.6	Initialization Vector Derivation.....	419
34	17.12.7	EC Public Key Point Encoding in Uncompressed Form	419
35	17.13	EXAMPLE	419
36	17.13.1	Keys	419
37	17.13.2	Message to encrypt	420
38	17.13.3	Encryption process	420
39	17.13.4	Decryption process.....	420
40	18	SECURITY	422
41	18.1	SPAKE2+ PROTOCOL DESCRIPTION.....	422
42	18.1.1	General.....	422

1	18.1.2	Execution.....	422
2	18.1.3	Verification.....	424
3	18.1.4	Summary.....	424
4	18.1.5	SPAKE2+ Constant Definitions.....	425
5	18.2	PRIVACY PROPERTIES	426
6	18.2.1	NFC Transaction [WCC1/WCC2].....	426
7	18.2.2	Vehicle OEM Server	426
8	18.2.3	User Privacy	426
9	18.2.4	Multi-Vehicle OEM Deployment	426
10	18.2.5	Error Handling	426
11	18.3	CRYPTOGRAPHIC ALGORITHMS.....	427
12	18.4	CRYPTOGRAPHIC PROTOCOLS	427
13	18.4.1	Server Password Generation	427
14	18.4.2	Vehicle-side public EC Point Generation	427
15	18.4.3	Device-side public EC Point Generation.....	427
16	18.4.4	Vehicle-side computation of Shared Secret.....	428
17	18.4.5	Device-side computation of Shared Secret	428
18	18.4.6	Derivation of Evidence Keys.....	428
19	18.4.7	Vehicle-side computation of Evidence.....	428
20	18.4.8	Device-side computation of Evidence	429
21	18.4.9	Derivation of System Keys [WCC1/WCC2/WCC3]	429
22	18.4.10	Generate Attestation Signature.....	429
23	18.4.11	Validate Attestation or Certificate	429
24	18.4.12	Secure Channel Command Encryption and Authentication	429
25	18.4.13	Secure Channel Response Encryption and Authentication	430
26	18.5	ERROR HANDLING [WCC1/WCC2/WCC3]	430
27	18.6	SECURE ELEMENT (SE).....	430
28	19	BLUETOOTH LOW ENERGY (LE) INTERFACE [WCC2/WCC3].....	432
29	19.1	BLUETOOTH LE FUNCTIONAL REQUIREMENTS	432
30	19.1.1	Vehicle	432
31	19.1.2	Device	432
32	19.2	BLUETOOTH LE PROCEDURES	433
33	19.2.1	Owner Pairing Connection Establishment.....	433
34	19.2.2	Bluetooth Encryption.....	445
35	19.2.3	Passive Entry: Bluetooth LE Setup:	445
36	19.3	DK MESSAGE FORMAT	449
37	19.3.1	UWB Ranging Service Message	452
38	19.3.2	SE Message	459
39	19.3.3	Supplementary Service Message - Time Sync	460
40	19.3.4	Supplementary Service Message - First Approach.....	462
41	19.3.5	Supplementary Service Message - RKE	463
42	19.3.6	Supplementary Service Message – UWB Final Data	464

1	19.3.7	Vehicle OEM App Message.....	465
2	19.3.8	Head Unit Pairing Message.....	465
3	19.3.9	DK Event Notification	468
4	19.4	TIME SYNCHRONIZATION.....	484
5	19.4.1	Definitions	484
6	19.4.2	General Description	487
7	19.4.3	Device Uncertainty	490
8	19.4.4	“Bluetooth LE Timesync” Procedures.....	490
9	19.4.5	“Bluetooth LE Timesync” message flow examples	493
10	19.4.6	Conditions for a “Bluetooth LE Timesync” procedure.....	495
11	19.5	DIGITAL KEY – FLOWS	496
12	19.5.1	Owner Pairing Flow.....	496
13	19.5.2	Head Unit Pairing Flow.....	506
14	19.5.3	Passive Entry.....	509
15	19.5.4	URSK Management	510
16	19.5.5	Flow Selection - Establish Secure Ranging	513
17	19.5.6	Standard Transaction over Bluetooth LE.....	519
18	19.5.7	Engine Start.....	521
19	19.5.8	Receiver - Sharing & First Approach	521
20	19.5.9	RKE and RKE-Hands-Free Function Flow	524
21	19.5.10	Bluetooth LE Activation Flow.....	536
22	19.6	DIGITAL KEY - PREFERENCE MANAGEMENT	537
23	19.6.1	Preference Management: Connected Vehicles < Connection Limit	537
24	19.6.2	Preference Management: Connected Vehicles >= Connection Limit	539
25	19.7	SUBEVENT HANDLING	539
26	19.7.1	Command Complete SubEvent Code Handling	539
27	19.7.2	Ranging Session Status Changed SubEvent.....	540
28	19.8	ERROR HANDLING.....	542
29	19.8.1	SE Transaction Recovery	542
30	20	UWB MAC AND CHANNEL ACCESS [WCC3].....	544
31	20.1	UWB MAC ARCHITECTURE	544
32	20.1.1	Overview.....	544
33	20.2	MAC TIME GRID.....	546
34	20.3	MAC TIME GRID SYNCHRONIZATION	552
35	20.4	HOPPING FLAG AND ROUND INDEX DETERMINATION	553
36	20.5	MAC PROTOCOL.....	555
37	20.5.1	Ranging Exchange Sequence	555
38	20.5.2	Ranging Session Setup.....	561
39	20.5.3	MAC Control Channel (Over Bluetooth LE)	564
40	20.5.4	UWB MAC Configuration	565
41	20.6	STS INDEX INCREMENTATION.....	567

1	20.6.1	Rules for STS Index Incrementation	568
2	20.6.2	STS Incrementation and Packet Mapping within a Ranging Round	569
3	20.6.3	Calculation of STS Index.....	569
4	21	UWB PHY [WCC3]	570
5	21.1	INTRODUCTION	570
6	21.2	UWB PHY BLOCK DIAGRAM	570
7	21.3	UWB PACKET FORMAT	570
8	21.4	UWB FRAME ELEMENTS	571
9	21.4.1	BPRF SYNC	572
10	21.4.2	BPRF SFD.....	573
11	21.4.3	Scrambled Timestamp Sequence (STS)	573
12	21.4.4	BPRF PHR.....	573
13	21.4.5	Standard Compliant Data Field	574
14	21.5	PULSE SHAPE COMBINATIONS	574
15	21.5.1	Symmetrical Root-Raised-Cosine Pulse – PulseShape 0x0	574
16	21.5.2	Precursor-Free Pulse – PulseShape 0x1	575
17	21.5.3	PulseShape Combinations.....	575
18	21.6	RANGING MARKER	576
19	21.7	UWB RF	576
20	21.7.1	Operating frequency bands and Channel assignments	576
21	21.7.2	Frequency Source Requirements	576
22	22	UWB SECURITY [WCC3]	578
23	22.1	CRYPTOGRAPHY	578
24	22.1.1	Deriving the STS Sequence from the DRBG	578
25	22.1.2	SP0 Frames Encryption.....	578
26	22.1.3	Key Derivation Functions (KDFs).....	579
27	22.1.4	UWB Tracking Prevention Between Ranging Recoveries.....	582
28	22.2	UWB RANGING.....	583
29	22.2.1	STS Index Management.....	583
30	22.3	UWB MODULE.....	584
31	23	REFERENCES.....	585
32		APPENDIX A.....	588
33	A.1.	Standard Transaction Cryptography Flow	588
34	A.2.	Fast Transaction Cryptography Flow	591
35	A.3.	Standard Transaction Cryptography Flow (with Reader Intent for Fast	
36		Transaction).....	593
37	A.4.	X.509 Schemes	597
38		APPENDIX B.....	603
39	B.1.	Instance Configurations	603
40	B.2.	Certificate Field Formats.....	603
41		APPENDIX C.....	608

1	C.1.	External CA Certificate	608
2	C.2.	Vehicle OEM Intermediate Certificate	609
3	C.3.	Vehicle OEM Key/Leaf Certificate	610
4	C.4.	Vehicle OEM Privacy Encryption Certificate	611
5	C.5.	Vehicle OEM Signature Verification Certificate	613
6	APPENDIX D.	OWNER PAIRING TEST VECTORS [TO BE UPDATED]	614
7	D.1.	Derivation of z_0, z_1 with Scrypt	614
8	D.2.	Computation of w_0, w_1	614
9	D.3.	Computation of L	615
10	D.4.	Computation of X	615
11	D.5.	Computation of Y	616
12	D.6.	Computation of Z, V (by device)	617
13	D.7.	Computation of Z, V (by vehicle)	619
14	D.8.	Derivation of K, CK, SK	620
15	D.9.	Derivation of Evidence Keys K_1, K_2	621
16	D.10.	Computation of Evidences M_1, M_2	621
17	D.11.	Derivation of System Keys	622
18	APPENDIX E.	NFC-F SUPPORT [WCC1]	623
19	E.1.	Introduction	623
20	E.2.	Device Requirement	623
21	E.3.	Preconditions	624
22	E.4.	Commands and responses	624
23	E.5.	Protocol Operation	626
24	APPENDIX F.	DIGITAL KEY FRAMEWORK API	629
25	F.1.	Overview	629
26	F.2.	Functional Requirements	629
27	APPENDIX G.		630
28	G.1.	Ranging Session Parameter Tables [WCC3]	630
29	G.2.	Hopping Sequence	631
30	G.3.	Default Hopping Sequence:	631
31	G.4.	AES-Based Hopping Sequence:	631
32	APPENDIX H.	[WCC3]	632
33	APPENDIX I.	[WCC3]	632
34	I.1.	PulseShape 0x0	633
35	I.2.	PulseShape 0x2	634
36	APPENDIX J.	[WCC3]	635
37	J.1.	UWB Test Vector	635
38	J.2.	UWB Test Vector without hopping	636
39	J.3.	UWB Test Vector with continuous hopping	641
40			

1 LIST OF FIGURES

2	Figure 2-5: Device Functional Elements.	62
3	Figure 2-7: Domain Versions	72
4	Figure 2-8: D-VS Agreement for Key Tracking	75
5	Figure 2-9: Version Agreement after Device Software Update	77
6	Figure 2-10: Example – manageKey() usage based on D-VS agreed version	78
7	Figure 2-11: V-OD-FW Agreement after Vehicle SW Update	79
8	Figure 4-1: Applet Instance Layout	99
9	Figure 4-2: Digital Key Object and Entitlements Attestation Package	100
10	Figure 4-3: Mailboxes Read/Write Permissions	103
11	Figure 6-1: Owner Pairing NFC Exchanges	133
12	Figure 6-2: Owner Pairing Flow-- Phase 0/1: Preparation/Initiation	134
13	Figure 6-4: SPAKE2+ Flow	138
14	Figure 6-5: Key Creation Data Transfer to Device	139
15	Figure 6-6: Key Creation Data Transfer to Device	140
16	Figure 6-7: Key Creation Info Retrieval by Vehicle	142
17	Figure 6-9: Error Management of Different Phases in Owner Pairing	153
18	Figure 6-11: Vehicle and Device Transaction Timeout	156
19	Figure 7-1: Standard Transaction Flow	160
20	Figure 8-1: Fast Transaction Flow	161
21	Figure 10-1: Check Presence Transaction	165
22	Figure 13-1: REV_100: Shared Key Termination in Vehicle	242
23	Figure 13-2: REV_100a: Shared Key Termination in Vehicle (Vehicle Required to be Online)	243
24	Figure 13-3: REV_110/120: Shared Key Termination in Owner/Receiver Vehicle OEM Account	244
25	Figure 13-4: REV_130a/140a: Shared Key Termination on Owner Device Natively/in Vehicle OEM App	245
26	Figure 13-5: REV_150: Shared Key Termination Based on Expiry Date of the Key	246
27	Figure 13-7: REV_160: Shared Key Termination by Device OEM (Device Security Issue)	248
28	Figure 13-8: REV_160a: Shared Key termination by Device OEM and Receiver device is Offline	249
29	Figure 13-9: REV_170: Shared Key Termination Due to Remote Wipe of Device	250
30	Figure 13-10: REV_200/210: Shared Key Termination on Receiver device Natively/in Vehicle OEM App	252
31	Figure 13-11: REV_220: Shared Key Termination Due to Local Wipe of Device	253
32	Figure 13-12: REV_300: Shared Key Suspension by Device OEM Account	254
33	Figure 13-13: REV_310: Shared Key Resume in Device OEM Account or on Device	255
34	Figure 13-14: REV_310a: Shared Key Resume Using ResumeAttestation	256
35	Figure 13-16: REV_400a: Owner Key Deletion in Vehicle UI (Change of Device)	258
36	Figure 13-17: REV_600: Owner key suspension due to device reported lost/stolen	260
37	Figure 13-18: REV_610: Owner key resumption after device reported lost/stolen	261
38	Figure 13-19: REV_700: Unpairing in Vehicle UI (Sale of Device)	262
39	Figure 13-20: REV_700a: Unpairing in Vehicle UI (Vehicle Required to be Online)	263
40	Figure 13-21: REV_710: Unpairing in Vehicle OEM App on Owner Device	264
41	Figure 14-1: Message Authentication and Privacy Encryption	272
42	Figure 14-2: Authentication and Privacy Encryption Certificate Chain	274

1	Figure 15-1: Authentication Command Flows Over Contactless Interface	285
2	Figure 15-2: Authentication Command Flows Over Wired Interface	285
3	Figure 16-1: Variant 1 Certification Chain Model	357
4	Figure 16-2: Variant 2 Certification Chain Model	358
5	Figure 19-1: Bluetooth LE Link Layer Connection Establishment.	434
6	Figure 19-2: L2CAP Connection-Oriented Channel.	434
7	Figure 19-3: Bluetooth LE Pairing and Encryption Setup.	436
8	Figure 19-4: Passive Entry Flow Diagram.	446
9	Figure 19-5: Connection performance in relationship with device, transmitter, and receiver requirements.	448
10	Figure 19-6: Synchronization Methods.	488
11	Figure 19-7: Synchronization state between device and anchor.	488
12	Figure 19-8: Time Synchronization Uncertainty Flow Diagram.	490
13	Figure 19-9: Successful Bluetooth LE timeSync at BT connection (Procedure 0).	491
14	Figure 19-10: Successful Bluetooth LE timeSync triggered by vehicle (Procedure 1).	492
15	Figure 19-11: Unsuccessful Timesync message.	493
16	Figure 19-12: Bluetooth LE Timesync message flow example 1.	493
17	Figure 19-13: Time Synchronization flow diagram example 2.	494
18	Figure 19-14: Bluetooth LE Time Sync message flow example 3.	495
19	Figure 19-15: Owner Pairing Flow for Bluetooth LE/UWB.	497
20	Figure 19-16: Bluetooth LE Secure OOB Pairing Prep.	501
21	Figure 19-17: Capability Exchange.	503
22	Figure 19-20: URSK Derivation Flow in a dedicated Standard Transaction	511
23	Figure 19-22: Flow Selection for Establishing Secure Ranging	514
24	Figure 19-23: Secure Ranging Setup Flow.	515
25	Figure 19-24: Sub-Optimal Flow.	516
26	Figure 19-25: Ranging Session State Machine.	517
27	Figure 19-26: Ranging Suspend Accepted Flow.	517
28	Figure 19-27: Ranging Suspend Delayed Flow.	518
29	Figure 19-28: Ranging Recovery Flow.	519
30	Figure 19-29: Standard transaction over Bluetooth LE.	520
31	Figure 19-31: First New Key Transaction Flow.	523
32	Figure 19-32: RKE Flow for an Event-based RKE Action.	526
33	Figure 19-33: RKE Flow for an Enduring RKE action with continuous confirmation.	528
34	Figure 19-34: Vehicle Function Status Retrieval and Event-based Update.	532
35	Figure 19-37: First Approach Activation.	537
36	Figure 19-38: Device Preference Management.	538
37	Figure 19-39: Required Capability Exchanged.	539
38	Figure 19-40: URSK Not Found.	540
39	Figure 19-42: Recovery Failed Flow.	542
40	Figure 19-43: URSK Refresh due to Status Word 6484 _h in CREATE RANGING KEY Response.	543
41	Figure 20-1: Digital Key RAN Concept.	545
42	Figure 20-2: Digital Key One-to-Many Ranging Protocol.	547
43	Figure 20-3: Overview of MAC Grid (See Table G- 1).	551
44	Figure 20-4: Overview of Block Synchronization Approach.	553

1	Figure 20-5: Example of UWB Message Flow for MAC Protocol.	556
2	Figure 20-6: Example of MAC Parameter Negotiation and Setting.	564
3	Figure 20-7: SP0 Packet Content.	565
4	Figure 20-8: Basic Principles of STS Incrementation.	568
5	Figure 21-1: Block diagram for IEEE 802.15.4z HRP UWB PHY.	570
6	Figure 21-2: SP3	571
7	Figure 21-3: SP0	571
8	Figure 21-4: PHR bit assignment.	573
9	Figure 21-5: Data Field Encoding for Standard PHY.	574
10	Figure 22-1: Overview of the KDF used and its relationship.	579

1 LIST OF TABLES

2	Table 2-1: Description of Links between Actors.....	47
3	Table 2-2: Visibility, Manageability, and Shareability properties of AccountRole.....	67
4	Table 2-3: Type number and AccountRole value mapping.	69
5	Table 2-4: Shareable types.....	69
6	Table 2-5: Entitlement association to AccountRole Types.....	70
7	Table 2-6: Owner Key Tracking Request for Re-Tracking in Full Migration.....	83
8	Table 2-7: Friend Key Tracking for Re-Tracking in Full Migration.	84
9	Table 2-8: Reception of Versioning Parameters by non-versioning capable Servers.....	90
10	Table 2-9: Active Version Table	90
11	Table 4-1: Constant values for Private Mailbox.....	105
12	Table 4-2: Signaling Bitmap Decoding	106
13	Table 4-3: Private Mailbox Content	107
14	Table 4-4: Confidential Mailbox Content.....	111
15	Table 4-5: Mailbox Access Rights.....	111
16	Table 5-1: Owner Pairing Command Set.....	112
17	Table 5-2: Generic Status Words.....	112
18	Table 5-3: Response to SELECT Command.....	113
19	Table 5-4: SPAKE2+ REQUEST Command Fields.....	114
20	Table 5-5: SPAKE2+ REQUEST Response Fields.....	114
21	Table 5-6: SPAKE2+_REQUEST Response Error Status Words	116
22	Table 5-7: SPAKE2+ VERIFY Command Fields	116
23	Table 5-8: SPAKE2+ VERIFY Response Fields.....	117
24	Table 5-9: SPAKE2+ VERIFY Response Error Status Words.....	117
25	Table 5-10: WRITE DATA Response Error Status Words	118
26	Table 5-11: Objects for Digital Key Creation.....	118
27	Table 5-12: Endpoint Configuration Data	120
28	Table 5-13: Mailbox Mapping.....	122
29	Table 5-14: Device Configuration Data.....	124
30	Table 5-15: GET DATA Response Error Status Words.....	126
31	Table 5-16: GET DATA Response Decrypted Payload for tag 7F20 _h	127
32	Table 5-17: GET DATA Response Decrypted Payload for tag 7F22 _h	127
33	Table 5-18: GET DATA Response Decrypted Payload for tag 7F24 _h	127
34	Table 5-19: GET DATA Response Decrypted Payload for Tag D3 _h	127
35	Table 5-20: GET RESPONSE Response Error Status Words	128
36	Table 5-21: OP CONTROL FLOW P1 Parameters.....	129
37	Table 5-22: OP CONTROL FLOW P2 Parameters for P1=10 _h (continue).....	129
38	Table 5-23: OP CONTROL FLOW P2 Parameters for P1=11 (end with success)	129
39	Table 5-24: OP CONTROL FLOW P2 Parameters for P1=12 (end with failure).....	129
40	Table 5-25: Certificates	130
41	Table 6-1: Error SW Condition List 1.	146
42	Table 6-2: Objects for Device Digital Key Certificate	147

1	Table 6-3: Owner Key Tracking Request	149
2	Table 6-4: Owner Key Tracking Response Parameters	150
3	Table 6-5: Recommended Minimum Vehicle and Device Timeout Values	157
4	Table 6-6: Pairing Password URL Syntax	158
5	Table 9-1: AUTH0 Command Transaction Type Coding.....	162
6	Table 9-2: User Authentication Policies.....	163
7	Table 11-1: Approved Sharing Methods.....	167
8	Table 11-2: content in JSON Format.....	172
9	Table 11-3: JSON formatted genericSharingData data structure.....	173
10	Table 11-4: sharingDataType enum Definition	175
11	Table 11-5: Key Creation Request.....	180
12	Table 11-6: Key Signing Request.....	186
13	Table 11-7: External CA Certificate Container	187
14	Table 11-8: Instance CA Certificate Container.....	187
15	Table 11-9: Endpoint Certificate Container.....	187
16	Table 11-10: Sharing Method Attestation Arbitrary Data to be Hashed and Signed.....	188
17	Table 11-11: Import Request.....	189
18	Table 11-12: Entitlement Data.....	190
19	Table 11-13: Attestation Package	192
20	Table 11-14: Online Sharing PIN Information Package.....	198
21	Table 11-15: Unencrypted Signed Online Sharing PIN Information Package	199
22	Table 11-16: Device PIN Entry Request	202
23	Table 11-17: Key Tracking and Online Attestation Delivery Request	203
24	Table 11-18: Encrypted Data Container for Owner Instance CA Certificate	204
25	Table 11-19: Key Tracking Response.....	204
26	Table 11-20: Key Configuration in TLV Format.....	207
27	Table 11-21: Standard Access Profiles.....	207
28	Table 11-22: Sharing Sec Info	209
29	Table 11-23: Unencrypted signed SharingSecInfo	210
30	Table 11-24: SBxD/KIS Intermediate Certificate Container.....	211
31	Table 11-25: SBxD/KIS Endpoint Certificate Container.....	211
32	Table 11-26: Error Code Message.....	216
33	Table 11-27: Error Codes	217
34	Table 12-1: serviceManagementRequest.....	227
35	Table 12-2: Service Management Request Data.....	227
36	Table 13-1: Vehicle-triggered Key Termination	237
37	Table 13-2: Device-triggered Key Termination.....	237
38	Table 13-3: Vehicle OEM-triggered Key Termination.....	238
39	Table 13-4: Device OEM-triggered Key Termination.....	238
40	Table 13-5: keyID usage for manageKey() with action=TERMINATE.....	239
41	Table 13-6: Detailed Key Termination Use Cases.....	240
42	Table 13-7: Resume Attestation	257
43	Table 13-8: User initiated Fleet Shared key termination scenarios	266
44	Table 13-9: Vehicle OEM Triggered Fleet Key Termination.....	266

1	Table 13-10: Device OEM Triggered Fleet Key Termination.....	266
2	Table 13-11: FMS Triggered Fleet Key Termination.....	267
3	Table 13-12: FMS Flow to Use Case Mapping	267
4	Table 14-1: Authentication and Privacy Keys	273
5	Table 14-2: Remote Termination Request Data Fields.....	276
6	Table 14-3: Remote Termination Request from Terminator Device	277
7	Table 14-4: Remote Termination Request from Vehicle OEM Server.....	278
8	Table 14-5: OEM App Data Attestation Input.....	280
9	Table 15-1: Command Availability on Interfaces.....	283
10	Table 15-2: Generic Status Words.....	287
11	Table 15-3: Coding of the Different Ranges of Class Byte Values	288
12	Table 15-4: INSTALL for INSTALL Content of Tag C9	288
13	Table 15-5: Applet Implementation Options	289
14	Table 15-6: Values for Protocol Data Type A (tag ‘86’)	290
15	Table 15-7: Values for Protocol Data Type B (tag 87 _h).....	291
16	Table 15-8: Status Word— Command successfully executed.....	291
17	Table 15-9: Coding of the notification.....	292
18	Table 15-10: Notification Context.....	292
19	Table 15-11: Value and Presence of the Different Fields in the HCI Notification Event.....	292
20	Table 15-12: SELECT Response Fields	295
21	Table 15-13: Endpoint Configuration.....	296
22	Table 15-14: Setting of Tag 46 _h and Tag 47 _h by vehicle for various wireless capability combinations.....	298
23	Table 15-15: Setting of Tag 46 _h and Tag 47 _h by the owner device for the key sharing.....	299
24	Table 15-16: Tag 47 _h indicating Key Class.....	299
25	Table 15-17: Endpoint verification information.....	300
26	Table 15-18: Endpoint Termination Request.....	304
27	Table 15-19: Endpoint Termination Attestation Data Fields.....	305
28	Table 15-20: Endpoint Termination Attestation	305
29	Table 15-21: Deletion Request	306
30	Table 15-22: AUTHORIZE ENDPOINT Command Payload.....	307
31	Table 15-23: AUTHORIZE ENDPOINT Internal Buffer Content Before Processing (offset 0)	311
32	Table 15-24: AUTHORIZE ENDPOINT Attestation Data Fields with P2 = 01 _h	312
33	Table 15-25: AUTHORIZE ENDPOINT Attestation Data Field with P2 = 03 _h	312
34	Table 15-26: AUTHORIZE ENDPOINT Internal Buffer Content After Processing (offset 0).....	312
35	Table 15-27: View Parameters	314
36	Table 15-28: View Endpoint Identifier Response in Internal Buffer.....	314
37	Table 15-29: View Instance CA Response in Internal Buffer	315
38	Table 15-30: AUTH0 P1, P2 Parameters.....	316
39	Table 15-31: AUTH0 Command Payload	316
40	Table 15-32: AUTH0 Response Payload.....	316
41	Table 15-33: AUTH1 Vehicle Authentication Data Fields.....	318
42	Table 15-34: AUTH1 Command Payload	318
43	Table 15-35: AUTH1 Response Payload Before Encryption	318
44	Table 15-36: Endpoint Authentication Data Fields	318

1	Table 15-37: PRESENCE0 Response Payload.....	320
2	Table 15-38: PRESENCE1 Command Payload.....	320
3	Table 15-39: PRESENCE1 Vehicle Authentication Data Fields.....	321
4	Table 15-40: PRESENCE1 Response Payload Before Encryption	321
5	Table 15-41: READ BUFFER Command Payload for Format 2	322
6	Table 15-42: Exchange Command Decrypted Payload	323
7	Table 15-43: Exchange Response Decrypted Payload	324
8	Table 15-44: CONTROL FLOW P1 Parameters	326
9	Table 15-45: CONTROL FLOW P2 Parameters	326
10	Table 15-46: CONTROL FLOW P1/P2 Values for Applet.....	326
11	Table 15-47: CREATE ENCRYPTION KEY Command Payload.....	328
12	Table 15-48: Encryption Key Attestation Data Fields.....	328
13	Table 15-49: Encryption Key Attestation.....	328
14	Table 15-50: GET PRIVATE DATA Command Payload.....	329
15	Table 15-51: SET PRIVATE DATA Command Payload.....	330
16	Table 15-52: SET CONFIDENTIAL DATA Command Payload	331
17	Table 15-53: SET CONFIDENTIAL DATA Internal Buffer Content Before Processing	331
18	Table 15-54: SETUP ENDPOINT Command Payload	332
19	Table 15-55: SETUP INSTANCE Command Payload.....	333
20	Table 15-56: SIGN Command Payload	334
21	Table 15-57: SIGN Command Payload	334
22	Table 15-58: SIGN Data Fields	335
23	Table 15-59: SIGN Response	335
24	Table 15-60: MANAGE UA Command Payload	336
25	Table 15-61: Delete Ranging Keys Request.....	337
26	Table 15-62: Endpoint Conversion Request.....	339
27	Table 15-63: CONVERT ENDPOINT Internal Buffer Content after Processing (offset 0).....	339
28	Table 15-64: Receiver Endpoint Key Attestation Signed by Sender	350
29	Table 15-65: Arbitrary Data Attestation.....	352
30	Table 17-1: migrationMetadata() Request	369
31	Table 17-2: migrationMetadata() Response: Status 200: OK.....	370
32	Table 17-3: recoverKeyData() Request	371
33	Table 17-4: recoverKeyData() Response: Status Code 200: OK.....	371
34	Table 17-5: trackKey() Request Body	374
35	Table 17-6: trackKey() Response: Status Code 200: OK	375
36	Table 17-7: inFleet() Request body	378
37	Table 17-8: inFleet() Response body: Status 200: OK	378
38	Table 17-9: registerKey() Request body.....	379
39	Table 17-10: registerKey() Response body: Status 200: OK.....	379
40	Table 17-11: preShare() Request Body.....	380
41	Table 17-12: preShareResponse Body: Status 200: OK.....	380
42	Table 17-13: preTrack() Request Body	381
43	Table 17-14: preTrack() Response Body: Status 200: OK	382
44	Table 17-15: manageKey() Request	383

1	Table 17-16: manageKey() Response: Status Code 200: OK.....	384
2	Table 17-17: versionUpdate() Request	385
3	Table 17-18: versionUpdate() Response: Status Code 200: OK.....	386
4	Table 17-19: eventNotification() Request	388
5	Table 17-20: eventNotification() Response: Status Code 200: OK	389
6	Table 17-21: infleetExternal() Request Body	392
7	Table 17-22: infleetExternal() Response body: Status 200: OK.....	392
8	Table 17-23: prepareKeySharingExternal() Request body	393
9	Table 17-24: prepareKeySharingExternal() Response Body: Status 200: OK	393
10	Table 17-25: manageKeyExternal() Request Body	394
11	Table 17-26: manageKeyExternal() Response Body: Status 200: OK	394
12	Table 17-27: eventNotificationExternal() Request Body	395
13	Table 17-28: eventNotificationExternal() Response body: Status 200: OK	395
14	Table 17-29: getKeyInformation() Request Body	396
15	Table 17-30: getKeyInformation() Response Body: Status 200: OK	396
16	Table 17-31: deFleet() Request Body	396
17	Table 17-32: deFleet() Response Body: Status 200: Ok.....	396
18	Table 17-33: cancelService() Request Body.....	397
19	Table 17-34: cancelService() Response Body: Status 200: OK.....	397
20	Table 17-35: HTTP Request Header	397
21	Table 17-36: ResponseHeader	398
22	Table 17-37: EncryptedDataContainer	399
23	Table 17-38: UnencryptedDeviceConfigurationData	401
24	Table 17-39: UnencryptedDeviceData	402
25	Table 17-40: keyData	402
26	Table 17-41: SharedAccountsData	403
27	Table 17-42: SharedAccount	404
28	Table 17-43: Shared Keys Data	405
29	Table 17-44: SharedKey	405
30	Table 17-45: UnencryptedMigrationMetadata	405
31	Table 17-46: UiBundle	406
32	Table 17-47: KeyInfo	406
33	Table 17-48: SharingInfo.....	407
34	Table 17-49: Entitlement	407
35	Table 17-50: SupportedEntitlement.....	408
36	Table 17-51: ActivationOptions	408
37	Table 17-52: SbxRemoteTerminationRequest.....	408
38	Table 17-53: GroupTermination.....	409
39	Table 17-54: KeyPair.....	409
40	Table 17-55: VehicleInfo.....	409
41	Table 17-56: KeyStatus	410
42	Table 17-57: DeviceType	410
43	Table 17-58: KeyOrigin.....	411
44	Table 17-59: Action.....	411

1	Table 17-60: ReasonType enum.....	411
2	Table 17-61: ReasonSubtype associated with ReasonType.....	411
3	Table 17-62: ActivationOption.....	412
4	Table 17-63: sharingPolicy.....	412
5	Table 17-64: preTrackDataType.....	412
6	Table 17-65: preShareDataType.....	413
7	Table 17-66: activationOptionPolicy.....	413
8	Table 17-67: Status Code Values.....	414
9	Table 17-68: Server SubStatus Codes	414
10	Table 19-1: AdvA field of ADV_IND.....	438
11	Table 19-2: AdvData field of ADV_IND.	438
12	Table 19-3: Definition of IntentConfiguration byte.....	438
13	Table 19-4: Mandatory fields in Pairing Request.	438
14	Table 19-5: Mandatory fields in Pairing Response.....	439
15	Table 19-6: DK Service UUID.	439
16	Table 19-7: SPSM Characteristic declaration.	440
17	Table 19-8: SPSM Characteristic value declaration.	440
18	Table 19-9: Vehicle SPSM and DK version Characteristic Declaration	440
19	Table 19-10: Vehicle SPSM and DK Version Characteristic Value Declaration	440
20	Table 19-11: Device Selected DK Version Characteristic Declaration	440
21	Table 19-12: Device Selected DK Version Characteristic Value Declaration.....	440
22	Table 19-13: Attribute Value Definition for Vehicle SPSM and DK Version Characteristic	441
23	Table 19-14: Device Selected DK Version Characteristic.....	441
24	Table 19-15: Supported Feature Definition Bitmap	443
25	Table 19-16 Vehicle Bluetooth Tx Power Level Characteristics.....	443
26	Table 19-17: Vehicle Antenna Characteristic declaration	444
27	Table 19-18: Vehicle Antenna Characteristic value declaration.....	445
28	Table 19-19: DK Message Format.....	449
29	Table 19-20: Message Header definition.	449
30	Table 19-21: Message Type definition.	449
31	Table 19-22: Payload Header definition.	450
32	Table 19-23: Message Type and its associated messages.	450
33	Table 19-24: Ranging_Capability_RQ message and its parameters.....	452
34	Table 19-25: Definition of the parameters for Ranging_Capability_RQ.....	452
35	Table 19-26: Ranging_Capability_RS message and its parameters.....	452
36	Table 19-27: Definition of the parameters for Ranging_Capability_RS.	453
37	Table 19-28: Ranging_Session_RQ message and its parameters.	453
38	Table 19-29: Definition of the parameters for Ranging_Session_RQ.	453
39	Table 19-30: Ranging_Session_RS message and its parameters.	454
40	Table 19-31: Definition of the parameters for Ranging_Session_RS.....	454
41	Table 19-32: Ranging_Session_Setup_RQ message and its parameters.	455
42	Table 19-33: Definition of the parameters for Ranging_Session_Setup_RQ.	456
43	Table 19-34: Ranging_Session_Setup_RS message and its parameters.....	457
44	Table 19-35: Definition of the parameters for Ranging_Session_Setup_RS.....	457

1	Table 19-36: Ranging_Suspend_RQ message and its parameter.....	457
2	Table 19-37: Definition of the parameter for Ranging_Suspend_RQ.	457
3	Table 19-38: Ranging_Suspend_RS message and its parameter.	458
4	Table 19-39: Definition of the parameter for Ranging_Suspend_RS.....	458
5	Table 19-40: Ranging_Recovery_RQ message and its parameter.....	458
6	Table 19-41: Definition of the parameter for Ranging_Recovery_RQ.....	458
7	Table 19-42:Ranging_Recovery_RS message and its parameter.	458
8	Table 19-43: Definition of the parameter for Ranging_Recovery_RS.	458
9	Table 19-44: Configurable_Ranging_Recovery_RQ message and its parameter.....	459
10	Table 19-45: Definition of the parameter for Configurable_Ranging_Recovery_RQ.....	459
11	Table 19-46: Configurable_Ranging_Recovery_Response message and its parameter.	459
12	Table 19-47: Definition of the parameter for Configurable_Ranging_Recovery_Response.....	459
13	Table 19-48: DK_APDU_RQ message and its parameter.	460
14	Table 19-49: Definition of the parameter for DK_APDU_RQ.....	460
15	Table 19-50: DK_APDU_RS message and its parameter.....	460
16	Table 19-51: Definition of the parameter for DK_APDU_RS.	460
17	Table 19-52: Time_Sync message and its parameters.	460
18	Table 19-53: Definition of the parameter for Time_Sync.	461
19	Table 19-54: First_Approach_RQ message and its parameters.....	462
20	Table 19-55: Definition of the parameter for First_Approach_RQ.	462
21	Table 19-56: First_Approach_RS message and its parameters.....	463
22	Table 19-57: Definition of the parameter for First_Approach_RS.....	463
23	Table 19-58: TLV carried in RKE_Auth_RQ	463
24	Table 19-59: TLV carried in RKE_Auth_RS	464
25	Table 19-60: UWB_Final_Data message and its parameters.	465
26	Table 19-61: Definition of the parameters for UWB_Final_Data.	465
27	Table 19-62: Pass_through message and its parameter.....	465
28	Table 19-63: Definition of the parameter for Pass_through.	465
29	Table 19-64: HU_PP message and its parameters.	466
30	Table 19-65: Definition of the parameter for HU_PP.....	466
31	Table 19-66: HUP_RQ message and its parameters.....	466
32	Table 19-67: Definition of the parameter for HUP_RQ.	466
33	Table 19-68: HUP_RS message and its parameters.....	467
34	Table 19-69: Definition of the parameter for HUP_RS.....	467
35	Table 19-70: DK Event Notification message and its parameters.	468
36	Table 19-71: Definition of the parameters for DK Event Notification.	468
37	Table 19-72: DK SubEvents Category and its parameters.....	468
38	Table 19-73: Definition of Command_Status and its Command_Status codes.	469
39	Table 19-74: List of Command_Status for Command Complete SubEvent.....	469
40	Table 19-75: Definition of Session_Status and its Session_Status codes.....	470
41	Table 19-76: List of Session_Status for Ranging Session Status Changed SubEvent.....	470
42	Table 19-77: Definition of DR_Intent and DR_Intent codes.....	471
43	Table 19-78: List of DR_Intent for Device Ranging Intent SubEvent.	471
44	Table 19-79: List of Vehicle Status Changed SubEvent Tags.....	472

1	Table 19-80: Source of Change values	476
2	Table 19-81: Definition of Function ids and their corresponding Action ids.	477
3	Table 19-82: Definition of Standardized Function/Execution Status Values to Indicate the (Temporary)	
4	Unavailability of Function/Execution or Errors.	480
5	Table 19-83: Definition of RKE Request SubEvent Templates.	481
6	Table 19-84: Definition of Headunit_Pairing_Status and its Session_Status.	482
7	Table 19-85: List of Session_Status for Head Unit SubEvent.	482
8	Table 19-86: List of UA Policies.....	482
9	Table 19-87: List of UA Policies.....	483
10	Table 19-88: PDR Data Request Parameters	483
11	Table 19-89: PDR Data Parameters.....	484
12	Table 19-90: 7F49 Template.....	499
13	Table 19-91: URSK storage requirements per Digital Key endpoint.	513
14	Table 19-92: RKE allowed user authentication configuration.....	524
15	Table 20-1: Example MAC Configuration.	551
16	Table 20-2: Mapping of Slots to UWB Ranging Packets.	556
17	Table 20-3: Pre-Poll Request Message and its parameters.	557
18	Table 20-4: The definition of parameters in the Pre-Poll Message.....	557
19	Table 20-5: Final_Data Message and its parameters.	559
20	Table 20-6: The definition of parameters in the Final_Data Message.	559
21	Table 20-7: Ranging Status of a Responder.	561
22	Table 20-8: The Contents of the MHR Field.....	565
23	Table 20-9: The Contents of the Frame Control Field.....	565
24	Table 20-10: The Contents of the Auxiliary Security Header.	566
25	Table 20-11: The Contents of the Vendor Specific Header IE.	567
26	Table 21-1: Supported UWB Configurations.	572
27	Table 21-2: The Mandatory Length-127 Ternary Code Sequences.....	572
28	Table 21-3: Summary of pulse shapes.....	575
29	Table 21-4: Overview of PulseShape_Combo values and the associated transmit pulse shapes.....	575
30		
31		

1 LIST OF LISTINGS

2	Listing 5-1: Vehicle Certificate Extension Schema.....	120
3	Listing 5-2: Vehicle Certificate Extension Data.....	120
4	Listing 5-3: Vehicle Public Key Certificate Data	121
5	Listing 11-1: Sample Provisioning Information in JSON Format	175
6	Listing 11-2: Sharing Invitation.....	185
7	Listing 11-3: Key Creation Processing.....	187
8	Listing 11-4: Import Request.....	190
9	Listing 11-5: Receiver, Endpoint Data Import Processing	193
10	Listing 11-6: First New Key Transaction	195
11	Listing 11-7: track key Response Handling.....	205
12	Listing 11-8: SBxD/KIS Endpoint Certificate Extension Data.....	211
13	Listing 11-9: SBxD/KIS Endpoint Certificate Extension Data.....	212
14	Listing 11-10: SBxD/KIS Endpoint Certificate Data	212
15	Listing 11-11: SBxD/KIS Intermediate CA Certificate Extension Schema.....	213
16	Listing 11-12: SBxD/KIS Intermediate CA Certificate Extension Data.....	214
17	Listing 11-13: SBxD/KIS Intermediate CA Certificate Data	214
18	Listing 11-14: setPrivateMailboxBit	217
19	Listing 11-15: clearPrivateMailboxBit	218
20	Listing 11-16: clearPrivateMailboxBytes.....	218
21	Listing 11-17:Generate Attestation Signature	218
22	Listing 14-1: Vehicle OEM Privacy Encryption Certificate Extension Schema	275
23	Listing 14-2: Vehicle OEM Privacy Encryption Certificate Extension Data	275
24	Listing 14-3: Vehicle OEM Signature Certificate Extension Schema.....	275
25	Listing 14-4: Vehicle OEM Signature Certificate Extension Data.....	275
26	Listing 14-5: OEM App Data Attestation Processing.....	280
27	Listing 15-1: INSTALL for INSTALL Processing.....	290
28	Listing 15-2: SELECT Processing.....	295
29	Listing 15-3: Endpoint Certificate Extension Schema.....	301
30	Listing 15-4: Endpoint Certificate Extension Data.....	301
31	Listing 15-5: Endpoint Certificate Data.....	302
32	Listing 15-6: CREATE ENDPOINT Processing.....	303
33	Listing 15-7: TERMINATE ENDPOINT Processing for Command Format 1.....	305
34	Listing 15-8: TERMINATE ENDPOINT Processing for Command Format 2.....	305
35	Listing 15-9: DELETE ENDPOINT Processing	306
36	Listing 15-10: External CA Certificate Extension Schema	307
37	Listing 15-11: External CA Certificate Extension Data	307
38	Listing 15-12: External CA Certificate Basic Constraints Extension Data.....	307
39	Listing 15-13: External CA Certificate Data	308
40	Listing 15-14: Instance CA Certificate Extension Schema.....	309
41	Listing 15-15: Instance CA Certificate Extension Data.....	309
42	Listing 15-16: Instance CA Certificate Data	310

1	Listing 15-17: AUTHORIZE ENDPOINT Processing.....	312
2	Listing 15-18: VIEW Processing.....	315
3	Listing 15-19: AUTH0 Processing	316
4	Listing 15-20: AUTH1 Processing	319
5	Listing 15-21: PRESENCE0 Processing	320
6	Listing 15-22: PRESENCE1 Processing	321
7	Listing 15-23: READ BUFFER Processing for Command Format 1.....	322
8	Listing 15-24: READ BUFFER Processing for Command Format 2.....	322
9	Listing 15-25: WRITE BUFFER Processing.....	322
10	Listing 15-26: EXCHANGE Processing	324
11	Listing 15-27: CONTROL FLOW Processing	327
12	Listing 15-28: CREATE ENCRYPTION KEY Processing.....	328
13	Listing 15-29: GET PRIVATE DATA Processing for Command Format 1	329
14	Listing 15-30: GET PRIVATE DATA Processing for Command Format 2	330
15	Listing 15-31: SET PRIVATE DATA Processing	330
16	Listing 15-32: SET CONFIDENTIAL DATA Processing	331
17	Listing 15-33: SETUP ENDPOINT Processing.....	333
18	Listing 15-34: SETUP INSTANCE Processing	334
19	Listing 15-35: SIGN Processing.....	335
20	Listing 15-36: onDeselect Event Processing	336
21	Listing 15-37: CREATE RANGING KEY processing.....	336
22	Listing 15-38: DELETE RANGING KEYS Processing.....	337
23	Listing 15-39: CONVERT Endpoint Processing 1.....	339
24	Listing 15-40: Generate Random.....	340
25	Listing 15-41: Generate Key Pair	340
26	Listing 15-42: Generate Identifier	340
27	Listing 15-43: Generate Attestation Signature	340
28	Listing 15-44: Verify Attestation Signature	341
29	Listing 15-45: Compute Shared Key with Diffie-Hellman.....	341
30	Listing 15-46: Key Derivation.....	341
31	Listing 15-47: Secure Channel Command Decryption and Authentication.....	341
32	Listing 15-48: Secure Channel Response Encryption and Authentication	342
33	Listing 15-49: Derive KEenc, KEmac	342
34	Listing 15-50: Confidential Mailbox Encryption and Authentication	342
35	Listing 15-51: Compute DeviceCryptogram	343
36	Listing 15-52: framework.createInstance Processing	344
37	Listing 15-53: framework.getInstance Processing.....	345
38	Listing 15-54: framework.view Processing.....	345
39	Listing 15-55: framework.deleteInstance Processing	345
40	Listing 15-56: instance.view Processing.....	346
41	Listing 15-57: instance.createEndpoint Processing	347
42	Listing 15-58: instance.deleteEndpoint Processing	347
43	Listing 15-59: instance.getEndpoint Processing.....	348
44	Listing 15-60: instance.setParameters Processing	348

1	Listing 15-61: endpoint.setParameters Processing	349
2	Listing 15-62: endpoint.getCertificate Processing.....	349
3	Listing 15-63: endpoint.terminate Processing	350
4	Listing 15-64: endpoint.authorize Processing.....	351
5	Listing 15-65: endpoint.createEncryptionKey Processing.....	351
6	Listing 15-66: endpoint.setConfidentialData Processing.....	351
7	Listing 15-67: endpoint.getPrivateData Processing.....	352
8	Listing 15-68: endpoint.setPrivateData Processing	352
9	Listing 15-69: endpoint.sign Processing.....	353
10	Listing 15-70: endpoint.auth0 Processing.....	353
11	Listing 15-71: endpoint.auth1 Processing.....	354
12	Listing 15-72: endpoint.exchange Processing	354
13	Listing 15-73: endpoint.createRangingKey processing.	354
14	Listing 17-1: Sample MigrationMetadata Request Body.....	370
15	Listing 17-2: Sample MigrationMetadata Response.....	370
16	Listing 17-3: Sample recoverKeyDataRequest Body	372
17	Listing 17-4: Sample recoverKeyData() Response.....	372
18	Listing 17-5: Unsuccessful Sample recoverKeyData() Response.....	374
19	Listing 17-6: Sample trackKey() Request Body.....	375
20	Listing 17-7: Sample trackKey() Response: Status: 200 OK.....	376
21	Listing 17-8: Sample preShare() Request	380
22	Listing 17-9: preTrack() Request.....	382
23	Listing 17-10: Sample manageKey() Request Body.....	385
24	Listing 17-11: Sample versionUpdate() Request Body	387
25	Listing 17-12: Sample versionUpdate() Response.....	387
26	Listing 17-13: Sample eventNotification() Request	390
27	Listing 18-1: Server Password Generation	427
28	Listing 18-2: Vehicle-side Public Point Generation	427
29	Listing 18-3: Device-side Public Point Generation	427
30	Listing 18-4: Vehicle-side Computation of Shared Secret	428
31	Listing 18-5: Device-side Computation of Shared Secret.....	428
32	Listing 18-6: Derivation of Evidence Keys	428
33	Listing 18-7: Vehicle-side Computation of Evidence.....	428
34	Listing 18-8: Device-side Computation of Evidence.....	429
35	Listing 18-9: Derivation of System Keys	429
36	Listing 18-10: Secure Channel Command Encryption and Authentication	429
37	Listing 18-11: Secure Channel Response Encryption and Authentication	430
38		
39		
40		

1 ABBREVIATIONS AND ACRONYMS

2	AES	Advanced Encryption Standard
3	AGC	Automatic Gain Control
4	AID	Application Identifier
5	APDU	Application Protocol Data Unit
6	Bluetooth LE	Bluetooth Low Energy
7	BPM	Burst Position Modulation
8	BPSK	Binary Phase Shift Keying
9	CA	Certificate Authority
10	CASD	Controlling Authority Security Domain
11	CMAC	Cipher-based Message Authentication Code
12	CoC	Connection Oriented Channel over Bluetooth LE
13	DK	Digital Key
14	DRBG	Deterministic Random Bit Generator
15	dUDSK	Derived UWB Data Secret Key
16	dURSK	Derived URSK
17	ECC	Elliptic Curve Cryptography
18	ECDSA	Elliptic Curve Digital Signature Algorithm
19	ECIES	Elliptic Curve Integrated Encryption Scheme
20	ERDEV	Enhanced Ranging Device
21	FMS	Fleet Management Server
22	FNKT	First New Key Transaction
23	GP	GlobalPlatform
24	HCE	Host Card Emulation; the NFC communication is routed to the framework instead
25		of the SE
26	HCI	Host Controller Interface
27	HRP	High Rate Pulse Repetition Frequency
28	HTTPS	Hypertext Transfer Protocol Secure
29	HU	Head Unit
30	ID&V	Identification and Verification (of User Identity)
31	KDF	Key Derivation Function
32	KIS	Key Issuing Server
33	KML	Key Management Logic
34	KTS	Key Tracking Server, operated by Vehicle OEM, privacy protected
35	L2CAP	Logical Link Control and Adaptation Protocol
36	Lc	Length Command

1	Le	Length Expected
2	LL	Link Layer
3	LSB	Least Significant Byte
4	MAC	Message Authentication Code
5	MFR	MAC Footer
6	MHR	MAC Header
7	MIC	Message Integrity Check
8	MITM	Man-in-the-Middle
9	MSB	Most Significant Byte
10	MSD	Message Sequence Diagram
11	mUPSK	Master UWB Privacy Secret Key
12	mURSK	Master URSK
13	NFC	Near Field Communication
14	NVM	Non-Volatile Memory
15	O2M	One-to-Many
16	OCSP	Online Certificate Status Protocol
17	OEM	Original Equipment Manufacturer
18	OOB	Out-of-Band
19	PAKE	Password Authenticated Key Exchange
20	PDR	Pedestrian Dead Reckoning
21	PHR	Physical Header
22	PHY	Physical Layer
23	PRF	Pulse Repetition Frequency
24	PSDU	Physical Layer Service Data Unit
25	PSM	Protocol/Service Multiplexer
26	QoS	Quality of Service
27	RAN	Ranging Area Network
28	RDEV	Ranging-capable device
29	RFU	Reserved for Future Use
30	RKE	Remote Keyless Entry
31	RS	Reed Solomon
32	RTR	Remote Termination Request
33	TA	Termination Attestation
34	SBOD	Server Based Owner Device
35	SBFD	Server Based Friend Device
36	SBxD	Server Based Owner Device or Server Based Friend Device
37	SCA	Side Channel Attack
38	SDS-TWR	Symmetric Double-Sided Two-Way Ranging

1	SE	Secure Element
2	SECDED	Single Error Correct Dual Error Detect
3	SFD	Start of Frame Delimiter
4	SHA	Secure Hash Algorithm
5	SHR	Synchronization Header
6	SP0	STS Packets type 0 (packets with payload and no STS)
7	SP3	STS Packets type 3 (packets without PHR, MHR, or payload)
8	SPAKE2+	Simple Password Authenticated Key Exchange
9	SPS	Service Provider Server
10	SPSM	Simplified Protocol/Service Multiplexer
11	STS	Scrambled Timestamp Sequence
12	SW	Status Word
13	SYNC	Synchronization Header
14	TLV	Tag Length Value
15	ToF	Time of Flight
16	TTL	Time to Live
17	UA	User Authentication
18	URL	Universal Resource Locator
19	URSK	UWB Ranging Secret Key
20	UWB	Ultra Wide Band

1 DEFINITIONS

2	Authorized User	A person or entity that has been authorized by the Vehicle Owner to
3		operate their Vehicle with a Digital Key for a certain Time Period
4		including the following examples: Friends, and family members, vehicle
5		rentals, and valet service.
6	Bluetooth Module	An entity managing the Bluetooth communication between device and
7		vehicle as described in this specification. This entity may consist of one or
8		more integrated circuits or may be part of a combo chip solution.
9	Bluetooth LE Pairing	The process for bonding two Bluetooth devices (a Vehicle and a Device)
10		so that the Bluetooth devices can automatically set up a secure connection
11		for future interactions. This is distinct from Owner Pairing, but can occur
12		during the Owner Pairing process.
13	Bluetooth LE Unpairing	The process where two Bluetooth unbond and remove the
14		information associated with the Bluetooth LE Pairing process, and the
15		devices are no longer able to automatically connect with each other.
16	Central	The master of the Bluetooth LE connection.
17	Delegation	The process of creating a key with sharing rights on a server or assigning
18		sharing rights to a server-based key for a specific vehicle. Delegation can
19		be performed independently of the existence of an owner key.
20	Device	A mobile device, tablet, or similarly functioning device that can wirelessly
21		communicate with a Vehicle directly or through a third-party application
22		to operate a Digital Key paired with the Vehicle.
23	Digital Key	A digital embodiment of the Physical Key used to operate a Vehicle. The
24		Digital Key is owned by the Vehicle Owner, who may authorize
25		‘Authorized Users’ to operate the Digital Key.
26	Digital Wallet	An application that the owner chooses that operates on a Device to store
27		the Digital Key.
28	Eligible User	Vehicle Owner and other eligible persons determined by the vehicle owner
29		who are allowed to perform a relevant action
30	Endpoint	Digital Key object in the applet.
31	Fleet Vehicle	A vehicle that has been in-fleeted and not yet-de-fleeted. This can be a
32		vehicle owned by a business entity or can be a private vehicle.
33	Initiator	An entity that starts the UWB ranging packet exchange by sending a first
34		UWB POLL packet.
35	Key Deletion	The process for removing a Digital Key from a Device.
36	Key Termination	The process of making a Digital Key no longer valid, and not accepted by
37		a Vehicle. This can involve deletion of the Digital Key from the device,
38		depending on the situation.

1	Key with sharing rights	Cryptographic key (pair) associated with rights to share and
2		manage certain other keys. Rights of the hared keys or persons to share to
3		may be limited.
4	Owner Device Pairing	The process of pairing of an Owner Device with a Vehicle. During
5		this process, a Vehicle and Device are authenticated to each other, and a
6		Digital Key is provisioned on the Owner Device. Owner pairing can be
7		accomplished using either NFC or Bluetooth link(s). Note that this is
8		distinct from Bluetooth LE Pairing. As per this specification, owner device
9		pairing deletes all shared keys.
10	Owner Device Swap	This optional but recommended process is accomplished by terminating
11		the current Owner Key and performing owner pairing with a second
12		device in any order. i.e., owner pairing with the second device may be
13		performed before or after the owner key on the first device is terminated.
14	Owner Key	Cryptographic key (pair) associated with a vehicle owner. This key has
15		sharing rights and cannot be deleted by other digital keys
16	Owner Key Termination	The process of terminating the Owner Digital Key on the Owner
17		Device, where it is no longer accepted as a valid Digital Key. Any
18		associated Friend Keys are maintained, unless explicitly terminated by the
19		user. This could be done when the owner replaces an old device with a
20		different one.
21	Owner Device Unpairing	The process of unpairing the Owner Device from the Vehicle that
22		returns the vehicle to a state identical to the state prior to Owner Pairing.
23		This process is described in Section 13.6 . This is distinct from Bluetooth
24		LE Unpairing
25	Peripheral	The slave in the Bluetooth LE connection.
26	PDR Data	Pedestrian Dead Reckoning data containing estimated position of the
27		Device based on Device's sensors like IMU (inertial measurement unit
28		using Accelerometer, Gyroscope and Magnetometer). See [45]
29	Physical Key	The dedicated physical key or key fob sold with the Vehicle used to
30		operate the Vehicle. The Physical Key is owned by the Vehicle Owner,
31		who may loan the Key to 'Authorized Users.'
32	Permanent Hardware Keys	Keys that have been purchased for the vehicle, which are bound to
33		non-mobile device hardware. These keys are not manageable via owner
34		device or other vehicle user interfaces. E.g., key fob and key card
35		implementations
36	Private Vehicle	A vehicle that has gone through owner pairing and at least one digital key
37		that origination through one or multiple P2P sharing steps from the owner
38		key still exists
39	Proof of Ownership	Secure method that might involve trusted artefacts used to prove the fact
40		of being the vehicle owner to a technical entity or a person
41	Responder	An entity that responds to a UWB POLL packet.

1	Shared Key	Digital Key shared with a receiver device. It is used interchangeably with
2		Receiver Digital Key.
3	Slot identifier	Value stored in private mailbox to reference a key slot of a Digital Key. Its
4		main purpose is to protect against re-use of deleted keys.
5	Time Period	The time-period that the Vehicle Owner has authorized the Authorized
6		user to have access to the Digital Key to operate their Vehicle.
7	UWB Module	An entity managing the UWB ranging session as described in this
8		specification. This entity may consist of one or more integrated circuits or
9		may be part of a combo chip solution.
10	Un-owned vehicle	Vehicle that is neither a private vehicle or a fleet vehicle (e.g., during
11		production, transport or at the dealership)
12	Vehicle	A vehicle that was complete and ready to use and had at least two (2)
13		wheels and physical controls enabling a driver to fully direct its speed and
14		direction over land (e.g., steering wheel, braking and acceleration pedals).
15		Notwithstanding the foregoing, “Vehicle” does not include any vehicle
16		that operate on water, in the air, or on rails.
17	Vehicle Binding	This step shall be performed prior to putting the vehicle into Owner
18		Pairing Mode and the creation of a Digital Key via the Owner Device
19		Pairing process. This process adds the vehicle into the Vehicle OEM
20		owner account and is Vehicle OEM proprietary and out of scope of this
21		specification.
22	Vehicle Defleeting	The process of deactivating or deleting an owner key on a server.
23	Vehicle Infleeting	The process of creating an owner key on a server or assigning owner
24		rights to a server-based key for a specific vehicle
25	Vehicle Unbinding	The process of removing vehicle from Vehicle OEM owner account.
26		During this process Digital Keys (owner and shared) associated with that
27		vehicle are terminated and no longer accepted as valid keys. This is
28		distinct from Owner Device Unpairing. This process is Vehicle OEM
29		proprietary and out of scope of this specification.
30	Vehicle Owner	A person or entity that owns the title to a Vehicle or the right to operate a
31		Vehicle including a vehicle lease.
32	Friend	An entity authorized by the owner to use the vehicle but with limited
33		entitlements. A friend may be a person or an organization. The term is
34		used to describe features and behavior when devices and vehicles defined
35		in this document interact with devices and vehicles as defined in [41].
36	Sender	An entity that shares a set of entitlements and capabilities using which a
37		recipient can create a Digital Key. The sender may be an owner or a
38		recipient of a shared key.

- | | | |
|---|----------|---|
| 1 | Receiver | The recipient of a set of entitlements and capabilities transmitted by the |
| 2 | | sender, that generates a Digital Key that is digitally signed by the sender |
| 3 | | and Vehicle OEM. The receiver may be a friend or an organization. |
| 4 | | |

NOTATIONS

Values like $A1_h$ or $FEED_h$ are hexadecimal values. Multi-octet integer values are encoded using the Most Significant Byte first convention (big-endian) unless explicitly stated to use Least Significant Byte first convention (little-endian).

The TLV fields are octets and shall be ordered as described.

Values like 1101_b are binary values using Most Significant Bit first convention.

Command and response APDU bytes and buffers are implicitly represented in hexadecimal notation.

Other numerical values like 1234 are decimal representations.

Fields in parentheses (..) are optional or conditional.

The concatenation operation is represented by $\parallel$.

ASCII values are represented by “quotes.”

String values shall be UTF-8 encoded unless specified differently.

Strings containing URLs/URIs shall be encoded as per RFC 3986 [27] unless specified differently.

Strings containing date values shall follow ISO-8601[2] definition using date format yyyy-MM-dd'T'HH:mm:ss.SSSZ unless specified differently

Binary data in JSON shall be encoded in hex string format (e.g., “deadbeef”) unless specified differently.

In fields described as a sequence of bits, bit7 is the most significant bit, bit0 is the least significant bit.

The notation K [start_index: number_of_bits] is used for keys.

The following table shows a list of the parameters/variables used in this specification for the purpose of the UWB MAC layer definition.

Parameter/Variable	Definition	Context
i	Ranging Block index	Ranging session
s	Ranging Round index	Ranging session
m	Slot index	Ranging session
k	Ranging Session index	Ranging session
N_{Round}^k	Number of ranging rounds in a ranging block	Ranging session
$\text{Round_Idx}^k(i)$	Ranging round index in ranging block i in the k -th ranging session that is used for the ranging exchange.	Ranging session

Parameter/Variable	Definition	Context
$N_{\text{Responder}}^k$	Number of responders	Ranging session
l	Index of responder	Ranging session
R_l	Responder with index $l=0,1,\dots, N_{\text{Responder}}^k - 1$	Ranging session
UWB_{time0}	Initiator time reference	RAN Global
UWB_{time0}^k	Initiator time reference for the k -th ranging session (by default, this defines the beginning of ranging block 1)	Ranging session
$UWB_{\text{time0}}^k(i)$	Initiator time reference for ranging block i for the k -th ranging session	Ranging session
$UWB_{\text{time0}}^k(i, \text{Round_Idx}^k(i))$	Initiator time reference for ranging exchanges in ranging block i , round $\text{Round_Idx}^k(i)$ for the k -th ranging session	Ranging session
STS_Index0^k	First STS index of the k -th ranging session, as negotiated during ranging session setup	Ranging session
$\text{STS_Index}^k(i)$	First STS index of ranging block i for the k -th ranging session	Ranging session
$\text{STS_Index}^k(i,s)$	First STS index of ranging round s of ranging block i for the k -th ranging session	Ranging session
$\text{STS_Index}^k(i,s,\text{TYPE})$	STS index of the slot whose type is TYPE in ranging round s of ranging block i for the k -th ranging session	Ranging session
T_{Round}^k	Ranging round duration	Ranging session
T_{Block}^k	Ranging block duration	Ranging session
$N_{\text{Slot_per_Round}}^k$	Number of slots in a ranging round	Ranging session

1
2
3

1 **CODE LISTINGS**

2 The pseudo-code listings presented in this document highlight the important processing steps
3 from a functional perspective. These pseudo-code listings do not represent any specific
4 implementation, but actual implementations shall be functionally equivalent. Operations like
5 basic input validation, length checking, input checking, option checking, or security counter-
6 measures may not be described. Memory or speed optimizations are not described in these
7 listings unless explicitly mentioned.

1 INTRODUCTION AND SCOPE

The Digital Key Technical Specification Release 3 specifies a Digital Key ecosystem using a standardized Digital Key applet, standardized vehicle access protocol, and a scalable architecture to support wide-scale deployment of Digital Keys across different Vehicle OEMs and Device OEMs. Release 3 is backward compatible with Release 2. Release 3 is not backward compatible with Release 1. Release 1 can be deployed independently of Release 2 or 3.

The Digital Key Technical Specification Release 3 is based on the use of BLE/UWB or NFC as an underlying radio technology to enable Digital Keys. The framework for Digital Key management is designed to be radio technologies agnostic, enabling the framework to be extended to support other technologies.

The Digital Key Technical Specification Release 4 enables multiple new features with the possibility for key sharing from servers-based devices and non-owner devices. Existing vehicles and devices can be upgraded individually in the field from any Release to Release 4. The use of versioning allows Release 4 entities to safely interoperate with Release 3 devices and vehicles.

NOTE: Beginning with Release 4, only online-generated, slot identifiers and (optional) immobilizer tokens are supported.

1.1 Release 4 Feature Summary

This new Release 4 of the Digital Key specification contains several new features relative to Release 3. Release 4 defines the method to share keys from senders to receivers in a way that allows the receiver to become a new sender and re-share the key another level, as defined by the new sender's account role. Senders can share keys based on this "sharing in a chain" feature in the Release 4 specification, but receivers might only support Release 3 features. Therefore, compatibility between Release 4 and Release 3 regarding sharing is achieved through versioning. The Release 4 specification also defines a new method for a second factor at sharing that vehicle OEMs can use to add additional security to sharing car keys, which shall be uniquely used for Release 4 to Release 4 sharing.

Besides sharing in a chain, the Release 4 Digital Key specification defines other features such as:

- SBOD
- Delegate key sharing
- Key classes
- Disable sharing

These features are described in Release 4 but can also be applied to vehicles and devices supporting Release 2 or Release 3 specifications only. Instead of making the editorial effort to integrate and maintain those features in all three releases of the Digital Key specifications, the Release 4 specification is meant to be the reference for all those revisions.

1.1.1 *Fleet Key Sharing*

The SBOD (server-based owner device) specification describes key management for fleet vehicles. It promotes an open standard that allows any CCC member company to build and operate an SBOD, it introduces a standardized infleeting API, it describes responsibilities for bringing and verifying the proof of ownership for fleet vehicles, proposes a simpler cross-signing scheme and will, in a future release, expand on a set of SBOD security requirements that should result in a CCC certification scheme for SBOD implementations that can be accepted by every vehicle OEM.

All elements of the SBOD architecture can be applied to vehicles and devices that support Release 2, Release 3 or Release 4 specifications equally.

1.1.2 *Delegated Key Sharing*

The delegate key sharing specification defines an SBFD (server-based friend device) together with several ways of authorizing sharing of delegate keys for owner-paired, in-fleeted and un-owned vehicles. Non-vehicle OEM-operated SBFDs require the reception of a delegate key that allows re-sharing. This authorization scheme mandates sharing in a chain as described in Release 4. Nonetheless authorization through owner signature or proprietary authorization methods work also for one-level sharing as defined in Release 2 and Release 3 specifications and can be applied to vehicles that are compatible with those releases.

1.1.3 *Key Classes*

The key classes feature allows key receiver devices or -servers to determine the identity of the sender device or server. This method is defined for multiple levels of re-sharing but can also be applied to one-level sharing. It is particularly required for sharing from servers to devices in order to distinguish device sharing from fleet or delegate sharing.

1.1.4 *Owner Sharing Control*

The possibility for the owner to disable sharing for their vehicles is required whenever there is a possibility for a server to issue keys for a specific vehicle. As SBOD and SBFD (with limitations) servers are specified to operate with Release 2, Release 3 and Release 4 vehicles, disabling of key sharing can and shall be used for all those vehicles.

1.1.5 *Summary*

Sharing in a chain as an extension of key sharing into multiple levels requires versioning to be introduced to the ecosystem of Release 2/Release3 and Release 4 compatible devices and servers. All other features of the Release 4 specification listed above can and should be applied to Release 2, Release 3 and Release 4 vehicles and devices equally as they are fully backwards compatible and provide new and useful features for various domains of digital car key usage.

1

2 SYSTEM ARCHITECTURE

Sections 2.1 through 2.7 are informative, and provide an overview of the features, components, architecture, and operations of the Digital Key system.

Sections 2.8 through 2.10, are normative.

2.1 Overview

The proposed system uses asymmetric cryptography to mutually authenticate vehicle and device. The device reveals its identity only to known vehicles.

Public keys are mutually exchanged through pairing of the owner device to the vehicle. The owner can then authorize the use of Digital Keys by friends and family members by signing their public keys.

The system is designed to work fully offline (i.e., no server connection is needed for a vehicle or a device) for all relevant features at the time of execution, such as the owner pairing or lock/unlock vehicle using the Digital Key. Where regulatory or business constraints require online connections, these can be added to the system.

2.2 High-Level Features

The described system complies with the following high-level requirements:

- Security and privacy equivalent to, or better than, physical keys
- Pairing of owner device and vehicle
- Key Sharing in a Chain (SiaC)
- Support of multiple Account Roles
- Sharing of keys for fleet vehicles generated on an SBOD through a Fleet Management Server (FMS)
- Sharing of keys from an SBFD through a Server Provider Server for delivery of vehicular services
- Issuing of replacement or emergency keys
- Interoperability across devices and Vehicle OEMs
- Support of multiple Digital Keys from different Vehicle OEMs on a single device
- Enabling of Vehicle OEM to control the issuance of Digital Key and its policy
- Privacy protection against active/passive eavesdroppers
- Infleeting and defleeting of vehicles managed by the fleet management server
- Conversion of existing R3 v.1.1 or higher Digital Keys into new Account Roles without re-provisioning

2.3 High level architecture

The Digital Key ecosystem consists of multiple actors that are connected to each other using a combination of standard and proprietary links as shown in [Figure 2-1](#). The standardized links are

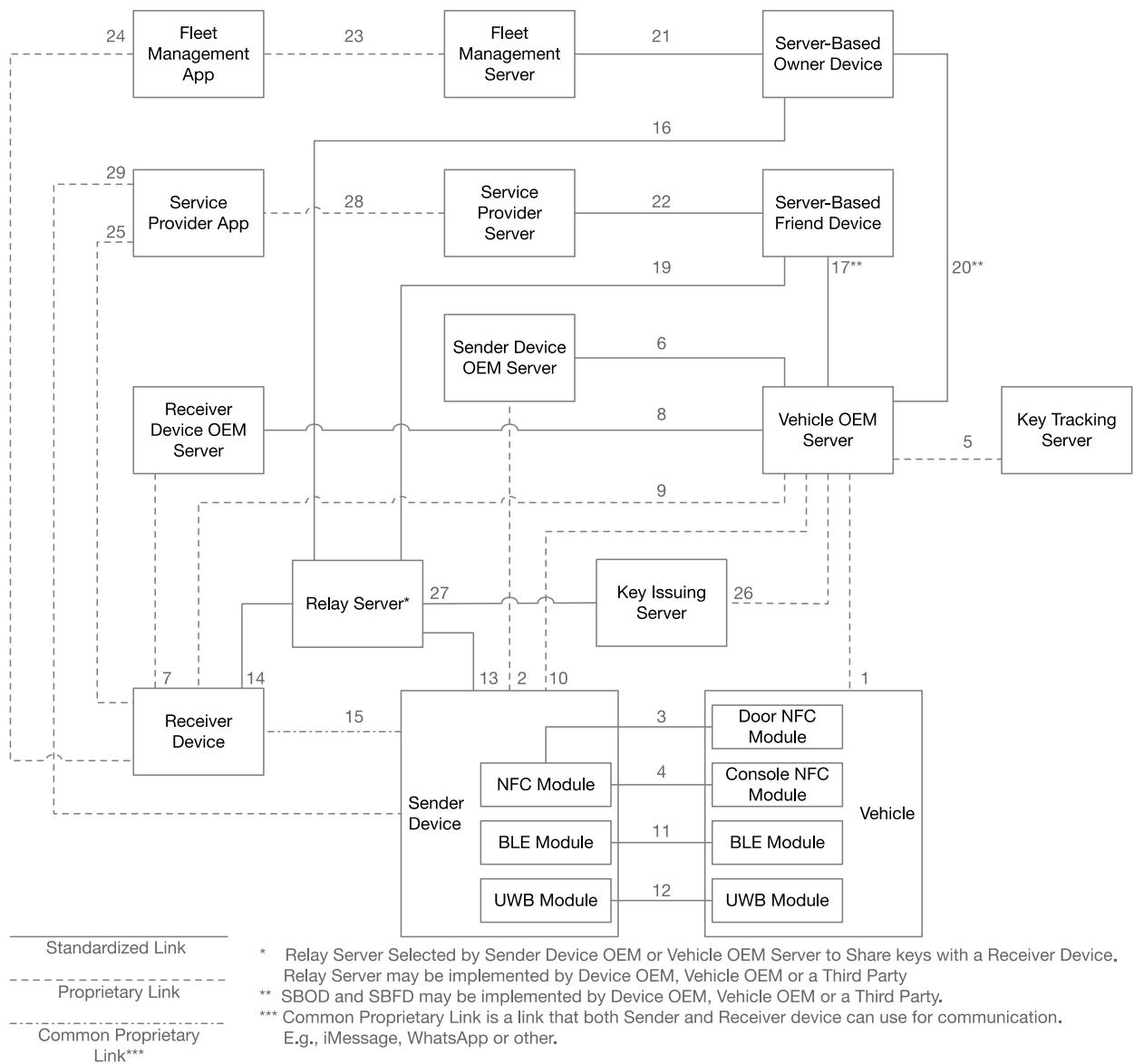
fully specified in this specification to enable implementation and interoperability. The roles and responsibilities of each entity and their relationships are described in Sections [2.4](#) and [2.5](#). [Figure 2-2](#) shows the architecture for private vehicles when not participating in fleet management or delegate services architectures. [Figure 2-3](#) shows the architecture for private vehicles when participating in a delegate services architecture. [Figure 2-4](#) describes the architecture for fleet vehicles when participating in a fleet services architecture (without delegation). The actors and relationships in [Figure 2-1](#) enable other architectures as well that are not shown here. For example, Fleet vehicles may participate in a delegated services architecture as well.

Note: The entities and the proprietary links (dashed lines in [Figure 2-1](#)) that are out of scope of this specification are also described to provide an overview of the high-level functionalities of the system.

Links between each actor in the digital key architecture are described in [Table 2-1](#)

1

Figure 2-1: Digital Key Architecture with Actors and their relationships



2

3

A detailed description of each link in [Figure 2-1](#) is provided in [Section 2.5](#)

4

5

Table 2-1: Description of Links between Actors

Link	Entities Connected	Standardized/Proprietary	Description
1	Vehicle and Vehicle OEM Server	Proprietary	Vehicle telematics link that provides a secure communication channel and is fully controlled by the Vehicle OEM
2	Sender Device and Sender Device OEM Server	Proprietary	Connects the device to the device OEM server. This link is used to manage Digital Key Lifecycle for sender or shared keys including processes to

Link	Entities Connected	Standardized/ Proprietary	Description
			suspend, restore and wipe digital keys when a device is lost or stolen.
3	Vehicle door NFC module and Device NFC module	Standardized	Used for door locking/unlocking and first transaction. See Section 2.5.1
4	Vehicle Console NFC module and Device NFC module	Standardized	User for Owner Pairing and Engine Start. Conducts first transaction for receiver devices. See Section 2.5.2
5	Vehicle OEM Server and Key Tracking Server	Proprietary	Link between Vehicle OEM Server and Key Tracking Server. The Key Tracking Server is used to register all Digital Keys and store related information using privacy preserving mechanisms
6	Vehicle OEM Server and Sender Device OEM Server	Standardized	Link which Device and Vehicle OEMs use to exchange server certificates and to create, manage, track and terminate digital keys.
7	Friend(or Receiver) Device and Friend(or Receiver) Device OEM Server	Proprietary	Connects the device to the device OEM server. This link is used to receive and manage the lifecycle of shared keys including processes to suspend, restore and wipe digital keys when a device is lost or stolen
8	Vehicle OEM Server and Receiver Device OEM Server	Standardized	Link which Receiver Device and Vehicle OEMs use to exchange server certificates and to create, manage, track and terminate shared digital keys
9	Receiver Device and Vehicle OEM Server	Proprietary	Direct link between Vehicle OEM Server and Vehicle OEM app residing on Receiver Device
10	Sender Device and Vehicle OEM Server	Proprietary	Direct link between Vehicle OEM Server and Vehicle OEM app residing on Sender Device
11	Vehicle BLE module and Device BLE module	Standardized	Used during Device to Vehicle initial key creation as well as Remote Key Entry functionality, UWB Ranging Session Setup and engine start
12	Vehicle UWB module and Device UWB module	Standardized	Used for ranging between Vehicle and Device
13/ 14	Relay Server and Sender, Receiver Device OEM Servers	Standardized	These links are used to enable secure key sharing between devices from the same device OEM or between devices from different device OEMs or between devices and the Key Issuing Server
15	Sender Device and Receiver Device	Common Proprietary Link	A known good communication channel over which a sender device and receiver can share information using which a receiver device can

Link	Entities Connected	Standardized/ Proprietary	Description
			generate a shared key. E.g., Messages, WhatsApp etc.
16	SBOD and Relay Server	Standardized	This link is similar to link 13/14. This link is used to securely share a digital key to a fleet user as described in [38].
17	SBFD and Vehicle OEM Server	Standardized or Proprietary	Used by the SBFD to track keys that have been shared to the SBFD. Also used by the Vehicle OEM Server to terminate keys on the SBFD when user discontinues a service. If SBFD is implemented by the Vehicle OEM, this link is regarded as proprietary.
18	Not Applicable	Not Applicable	Unused
19	SBFD and Relay Server	Standardized	This link is used to receive a key from a sender device to enable a delegate service. The SBFD also uses this link to share a delegate key.
20	SBOD and Vehicle OEM Server	Standardized or Proprietary	Used by the SBOD to in-fleet and de-fleet vehicles and deliver proof of ownership, if required. This link is also for digital key lifecycle management. If SBOD is implemented by the Vehicle OEM, this link is regarded as proprietary
21	Fleet Management Server and SBOD	Standardized	This link is used to start in-fleeting or de-fleeting and to initiate key sharing to a fleet user as well as to manage and transmit notifications relevant to key life-cycle management.
22	Service Provider Server and SBFD	Standardized	Link used to initiate sharing of a service key and to deliver success or failure information. Can also be used to suspend or terminate a digital key.
23,24	Fleet Management Server and Receiver Device	Proprietary	Links 23 and 24 connect the Fleet Management Server to the Receiver device via the Fleet Management App. The interface between the Fleet Management App and the Receiver Device is proprietary.

Link	Entities Connected	Standardized/ Proprietary	Description
25, 28, 29	Service Provider Server and Receiver Device	Proprietary	Links 25, 28 and 29 connect the Service Provider Server to the Sender/Receiver device via the Service Provider App. The Service Provider App is an entity that can be implemented using various mechanisms. The link between the Service Provider App and the receiver device is proprietary. This set of links allows the following: • sender device can send an invite to share a key to a SBFD • sender device to request delegate services, for which a key needs to be shared to a receiver device, • receiver device to receive a key sharing link from the service provider server.
26	Vehicle OEM Server and Key Issuing Server	Proprietary	Link used by the Key Issuing server to generate keys for vehicles that may not have gone through the owner pairing process (e.g., during production). These keys may take the form of NFC Keycards, key-fobs etc.
27	Key Issuing Server and Relay Server	Standardized	In some situations (e.g., no other keys available) the Key Issuing Server is capable of delivering a key to a receiver device using this link

A Vehicle OEM operated SBFD can additionally be enabled via proprietary methods or via a standardized Digital Key signing command. The Digital Key signing command is described in [15.3.2.23](#).

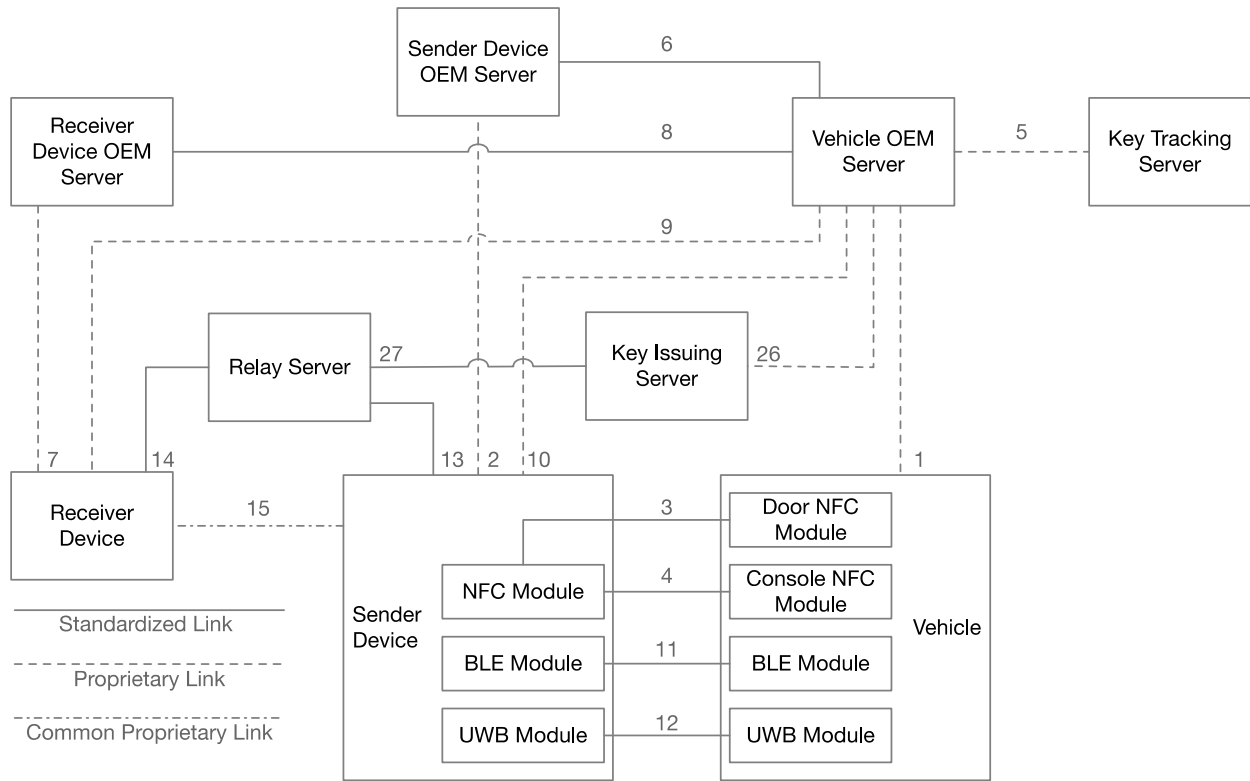
If a Vehicle OEM operated SBxD provides appropriate proof of ownership mechanisms and sharing control for the owner in place, then the Vehicle OEM operated SBxD may issue keys for a vehicle that has not yet performed owner pairing.

The **device** can share a Digital Key with another **device** (via (2), (6), (8) and (7)), defining the appropriate access profile and can terminate the shared Digital Key when it is no longer needed. Key sharing can be accomplished either via (2) and (7), (most commonly when the sender Device and receiver device are manufactured by the same OEM), or through a Relay Server (most commonly when the sender Device and receiver Device are manufactured by different OEMs: via (13), (14)). The invitation to obtain a shared key from the relay server is sent either from the sender device to the receiver device via (15). The invitation may use a second factor authentication scheme where the receiver proves its identity using schemes defined and required by either Device OEMs (such as Device PIN) or Vehicle OEMs as outlined in Section [11.2.2](#).

The **receiver device** and its **receiver Device OEM Server** connection (7) support necessary certificate services, as in link (2).

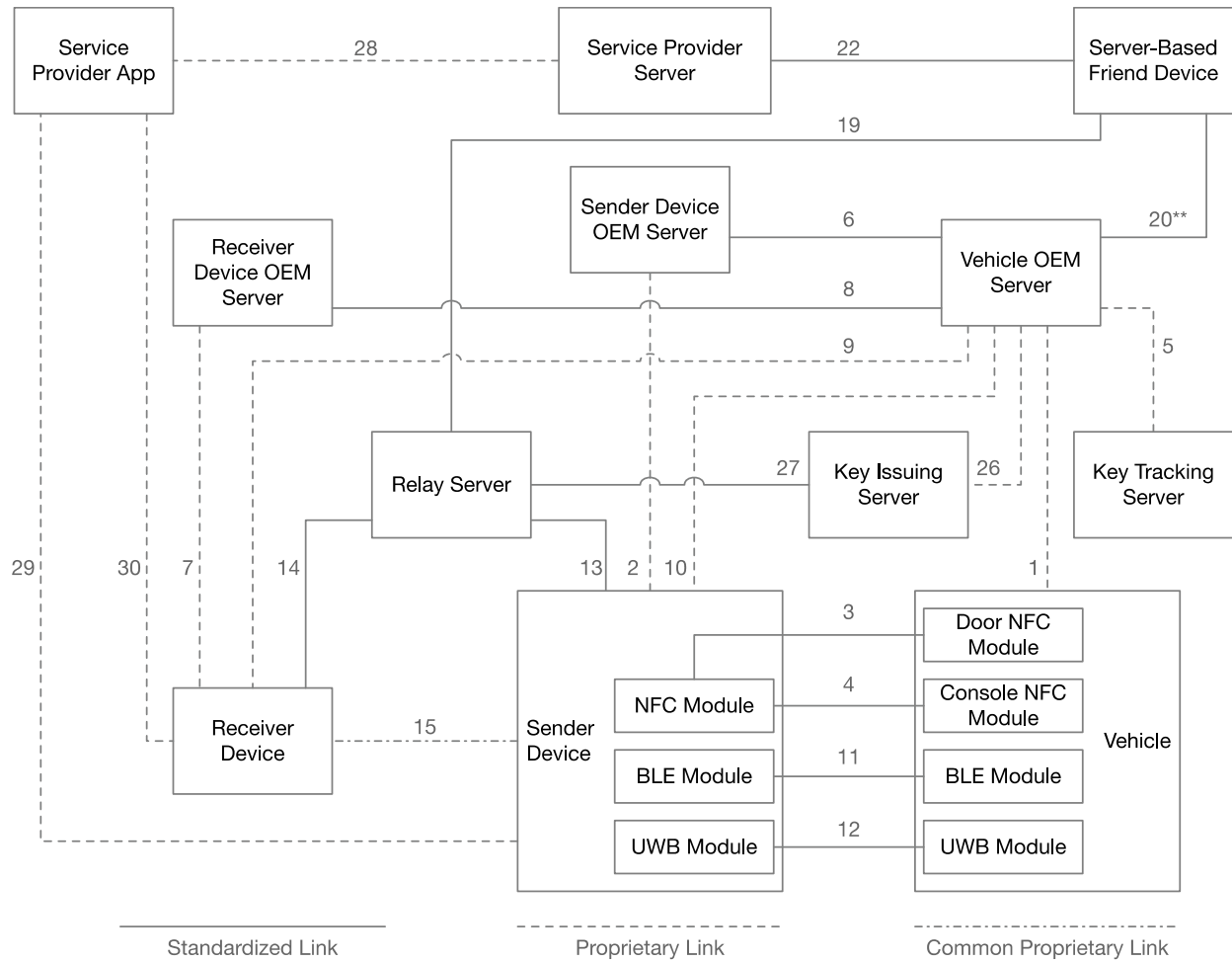
All eligible devices contain a certified SE as well as NFC, BLE or UWB capability to enable them to communicate with the vehicle.

Figure 2-2: Digital Key Architecture for Private Vehicles (without delegation)



1

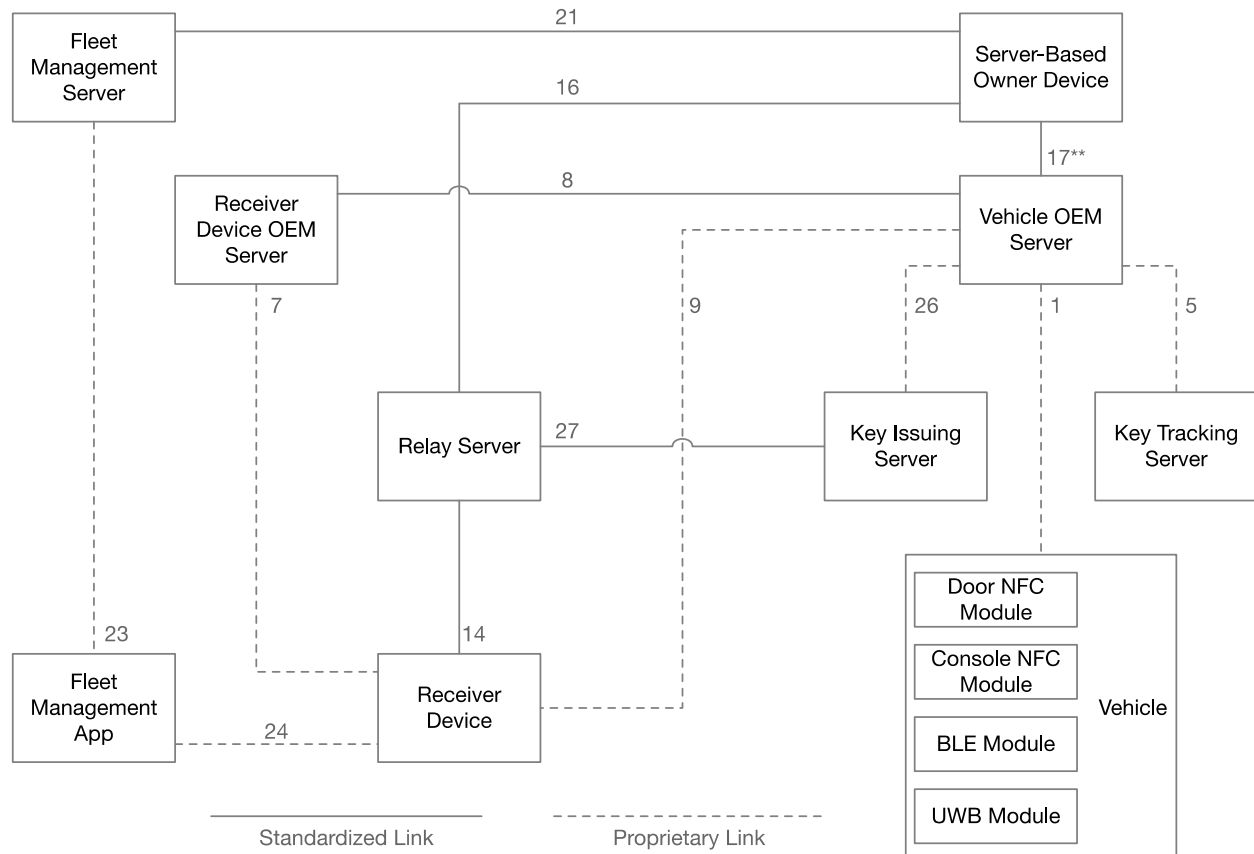
Figure 2-3: Digital Key Architecture for Private Vehicles (with delegation)



2

1

Figure 2-4: Fleet Digital Key for Fleet Vehicles without Delegation



2

3

4 2.4 Actors

5 In this section, the roles and responsibilities of the entities involved in the Digital Key ecosystem
 6 are described. The subsections within 2.4 describe the primary functions of the various actors.

7 2.4.1 Vehicle

- 8 • Determine whether a device is eligible for Digital Keys before allowing owner pairing or
- 9 accepting keys shared by the owner device, other eligible devices, SBOD or SBFD.
- 10 • May provide owner pairing information (owner public key, device information, etc.) and
- 11 shared Digital Key information to the KTS or verify that owner pairing and sharing
- 12 information was received by the KTS.
- 13 • Verify authenticity of the device.
- 14 • Authorize any device that can prove that it has a valid Digital Key to access the vehicle, and
- 15 if required by the vehicle, an immobilizer token to start the engine.
- 16 • Provide a user interface to delete owner and shared Digital Keys.
- 17 • Provide a method to control sharing to the owner and other eligible devices.
- 18 • Provide secure processing and storage environment.

2.4.2 *Vehicle NFC Readers [WCC1]*

- Communicate with the owner device for owner pairing and Digital Key transactions (lock/unlock, engine start, etc.).
- Communicate with the receiver devices for Digital Key transactions.

2.4.3 *Vehicle Bluetooth LE (BLE) Module [WCC2/WCC3]*

- Communicate with the BLE module of the owner device for owner pairing and Digital Key transactions (lock/unlock, engine start, RKE etc.).
- Communicate with the BLE module of the receiver device for “First New Key Transaction” and Digital Key transactions
- Communicate with the BLE module of the key holder device for setup of a secure ranging over UWB
- Communicate with the BLE module of owner or receiver devices for Remote transactions to allow the device to initiate on-demand features (e.g. Lock/Unlock etc.)
- Communicate with the BLE module of owner or receiver devices for transmission of Notifications to indicate change of state information
- Communicate with the BLE module of owner or receiver devices for transmission of 3rd party vehicle OEM application data.

2.4.4 *Vehicle UWB Module [WCC3]*

- Communicate with the UWB module of owner or receiver device for secure ranging to securely determine the distance between device and vehicle to support a secure distance measurement for passive entry and passive engine start functionality.

2.4.5 *Vehicle OEM Server*

- Host owner account that links to the owner’s vehicle(s) and manages ID&V.
- The **Vehicle OEM Server** hosts user accounts.
- Manage Digital Key subscriptions.
- Sign a shared Digital Key structure for acceptance by vehicle, assuring that business policies are checked and Digital Keys are tracked.
- Provide necessary attestations to the vehicle (when online) so that shared Digital Keys are accepted by the vehicle in the “First New Key Transaction”.
- Terminate Digital Keys in the vehicle when they have been deleted on a device.
- Synchronize with the device to perform offline termination of Digital Keys.
- Manage a secure channel to the vehicle.
- Create pairing passwords and provide them to owner device and vehicle.
- Sign vehicle public key.
- Provide necessary certificates to Device OEMs.
- Provide Vehicle OEM and (optionally) Device OEM public keys to vehicle for owner pairing and key sharing.

- Send data structures, uiBundle([Table 17-46](#)) and sharedAccountData([Table 17-41](#))
- Provide differentiation between fleet vehicles and private customer vehicles
- Can implement SBOD
- Can implement SBFD
- Can send digital key specific UI information to the device if vehicle OEM implements SBOD/SBFD/KIS
- Managing unlinking (defleeting) of the SBOD from the vehicle

2.4.6 Key Tracking Server (KTS)

- Record relevant data to be able to assign a tracked Digital Key for a vehicle to a device. The KTS is likely to be managed by the Vehicle OEM.
 - Verifies anonymized account information for secure intra-account sharing
- The following are the key properties of the KTS:
- It is only accessed for data query when required for legal or insurance reasons or for transmission of receiver key information during owner device change (see [Section 13.5.1.1](#))
 - Maintain data separation from the Vehicle OEM Server to fulfill privacy requirements of tracked data

2.4.7 Devices

- Contain a secure processing and storage environment (SE or equivalent) running a Digital Key applet
- Can take on the role of an owner device, receiver device or sender device
- Support contactless transactions to lock/unlock vehicle and start the engine
- Support configurable user authentication (e.g., passcode)
- Check service eligibility before allowing owner pairing or acceptance of a shared Digital Key

2.4.7.1 Owner Device

- Implement the main features of transaction, owner pairing, Digital Key sharing (sender), and Digital Key termination
- Store necessary certificates for owner pairing and Digital Key sharing
- Terminate shared Digital Keys by sending termination request to vehicle (via Vehicle OEM Server) and to receiver device via Device OEM Server and Vehicle OEM Server
- Provide a common proprietary link (15) from [Figure 2-1](#) for Digital Key sharing.

2.4.7.2 Receiver Device

- Implement the main features of transaction, Digital Key sharing (receiver), Digital Key termination
- Store necessary certificates for Digital Key sharing
- Send termination attestation to Vehicle OEM Server
- Provide a common proprietary link (15) from [Figure 2-1](#) to accept shared Digital Keys

2.4.7.3 *Sender Device*

- Implement main features: transaction, Digital Key sharing (sender), Digital Key termination
- Store necessary certificates for Digital Key sharing
- Terminate shared Digital Keys by sending termination request to vehicle (via Vehicle OEM Server) and to receiver device via Device OEM Server and Vehicle OEM Server
- Provide a common proprietary link (15) from [Figure 2-1](#) for Digital Key sharing

2.4.8 *Device OEM Server*

- Load and install the Digital Key instance of the Digital Key applet (if necessary)
- Provide and update necessary certificates in the device
- Allow Digital Key functionality to be temporarily disabled on a lost or stolen device (if online)
- Allow wipe of Digital Keys on a lost or stolen device (if online)

2.4.9 *Relay Server*

- Provides a standardized communication channel to support key sharing between two devices from different (or the same) OEMs. The standardized communication channel is outlined in the Secure Credential Transfer RFC [\[38\]](#).
- Implements push notifications and supports polling mechanisms to keep devices informed during the key sharing process.
- May be implemented by any entity and must be included in the CCC-approved Relay Server providers listed in [\[35\]](#)

2.4.10 *Server Based Owner Device (SBOD)*

- Generates and securely stores the owner Digital Keys for fleet vehicles.
- Provides APIs to in-fleet and de-fleet vehicles
- Manages the Digital Key lifecycle for fleet vehicles of behalf of the Fleet Management Server (FMS)
- Terminates owner Digital Key when requested by FMS or Vehicle OEM server.

2.4.11 *Server Based Friend Device (SBFD)*

- Hosts delegate service keys for private and fleet vehicles.
- Shares Digital Keys to delegate key user device and manages the receiver digital key lifecycle.
- Receives shared delegate keys.
- When implemented by the Vehicle OEM, the receiver digital key that is hosted on the SBFD, may be generated by both proprietary and standardized means.
- Generates and securely stores the signing key.

2.4.12 *Fleet Management Server (FMS)*

The following are the key functions of the FMS:

- Receives and manages infleeted vehicle(s)..
- Enrolls vehicle user to the fleet management service.
- Manages access to Digital Keys.
- When a user registers with the FMS through the Fleet Management App (2.4.13), the FMS binds the account identifier and device.
- Initiates sharing of digital keys of infleeted vehicles to vehicle user, device and/or device of fleet owner staff personnel.
- Terminates receiver Digital Keys
- De-fleets the vehicle when the vehicle leaves the fleet.
- Receives event notifications and key sharing data from SBOD during addition or removal of any shared keys.

2.4.13 *Fleet Management App*

- The Fleet Management App is the user interface to the FMS and may reside on a fleet user device.
- Communicates with the fleet user device of the customer or fleet owner personnel.
- When a user registers with the FMS, the Fleet Management App assists the FMS in binding the account identifier and the device.
- The fleet Management App may be used to obtain a URL using which the receiver device can obtain a fleet key for the fleet vehicle.

2.4.14 *Key Issuing Server (KIS)*

- Can issue keys when the vehicle is unpaired (e.g., during production phase) and manage key lifecycle.
- Can issue keys when the vehicle is paired (e.g., emergency keys)
- Can share replacement keys (e.g., keyfobs, NFC card, etc.)
- Can initiate a factory reset of the digital key system of the vehicle via the vehicle OEM server.

2.4.15 *Service Provider Server (SPS)*

- Enrolls the vehicle user for services offered by the Service Provider.
- Manages access to Digital Keys
- When a user registers for a service through the Service Provider App (Section [2.4.16](#)), the SPS binds the account identifier and the user device.
- Initiates delegate Digital Key sharing to a vehicle user, device and/or device of service provider personnel.

- Manages receiving and sending the sharing URL for service activation and service use.
- Terminates Digital Keys
- Receives event notifications and key sharing data from SBFD during addition or removal of any shared keys.
-

2.4.16 Service Provider App

- The Service Provider App is the user interface to the SPS and may reside on a private user or fleet user device.
- Communicates with a vehicle user, device and/or device of Service Provider personnel for user activated services offered by the Service Provider
- When a user registers for a service provided through the SPS, the Service Provider App assists the SPS in binding the account identifier and the device.

2.5 Relationships

In this section, the relationships between different entities are described. Links referred to in this section (in parentheses) are shown in [Figure 2-1](#).

The following links are standardized according to the protocol, and relevant message formats are defined in this specification..

- **Keyholder Device:** Door NFC reader (link 3) as defined in Section [2.5.1](#)
- **Keyholder Device:** Console NFC reader (link 4) as defined in Section [2.5.2](#)
- **Keyholder Device OEM Server:** Vehicle OEM Server (links 6 and 8) as defined in Section [2.5.10](#).
- **Keyholder Device:** Bluetooth LE Interface (link 11) as defined in Section [2.5.3](#).
- **Keyholder Device:** UWB Interface (link 12) as defined in Section [2.5.4](#).
- **Relay server:** Sender and receiver device (links 13 and 14) as defined in Section [11.3](#). [SBOD \(link 16\)](#), [SBFD \(link 19\)](#) and [KIS \(link 27\)](#) as defined in Section [12](#).
- **FMS/Service Provider Server:** SBOD (link 21) and SBFD link (22) as defined in Section [12](#).
- **SBOD/SBFD:** Vehicle OEM Server (link 17/20) as defined in Section [12](#).

All other links are out of scope of this specification and are described at a high level to provide an overview of the underlying links and their functionality. Standardized information may be transported across out-of-scope links.

2.5.1 Door NFC Reader (3) [WCC1]

- Conducts regular transactions (fast or standard) with all devices that have a valid and registered Digital Key
- Conducts first transaction for a receiver device during Key Sharing, by transmitting necessary attestations so that the vehicle can verify the shared Digital Key.
- Implements specific polling to allow automatic selection of the appropriate Digital Key

2.5.2 Console NFC Reader (4) [WCC1]

- Conducts (standard or fast) engine start transactions with all devices that have a valid and registered Digital Key
- Authorizes engine start
- Conducts owner pairing transaction with a device to enable it to become the owner device
- Conducts first transaction for a receiver device in cases when the vehicle was offline during key sharing, by transmitting necessary attestations so that the vehicle can verify the shared Digital Key.
- Implements specific polling to allow automatic selection of the appropriate Digital Key (see Section [2.10](#), NFC Interface)

2.5.3 Bluetooth LE Interface (11) [WCC2/WCC3]

- Supports standard transactions with all devices that have a valid and registered Digital Key
- Supports owner pairing transaction with a device to enable it to become the owner device
- Supports first transaction for a receiver device
- Supports setup of the secure UWB ranging session
- Supports Remote transactions to allow the device to initiate on-demand features (e.g. Lock/Unlock etc.)
- Supports Transmission of Notifications to notify and signal change of state information
- Supports Transmission of data of 3rd party vehicle OEM application.

2.5.4 UWB Interface (12) [WCC3]

- Supports secure ranging operation to determine the distance between device and vehicle in support of secure distance measurement for passive entry and passive engine start functionality.

2.5.5 Sender-to-Receiver Device Link (via (2), (6), (8), and (7))

2.5.5.1 Sender-to-Receiver Device Link (via (2), (6), (8), and (7))

- Owner (or sender) device deletes Digital Keys from the receiver device by sending termination and deletion commands
- Receiver device sends termination attestation when a Digital Key on the device is terminated

2.5.5.2 Sender-to-Receiver Device Link (via (13), (14) and (15))

- Sender device shares Digital Keys with receiver device using sequence of communications links: (15), (13), (14)

2.5.6 Sender or Receiver Device to Vehicle OEM Server (10, 9)

- Links 9 and 10 are realized through the Vehicle OEM app; see Section [2.6.7](#)

2.5.7 *Telematics Link (1)*

- Proprietary, trusted, and confidentiality-preserving link that is managed by the Vehicle OEM. Key functions include:
 - Send owner pairing verifier and additional pairing information to the vehicle
 - Obtain signature of the vehicle public key from the Vehicle OEM CA
 - Send Digital Key termination information to delete a specific Digital Key in the vehicle
 - Notify Vehicle OEM Server about Digital Key deletion in vehicle to remove Digital Key from the receiver device
 - Register owner Digital Keys with Vehicle OEM Server (KTS)

2.5.8 *Sender/Receiver Device OEM Server Link (2,7)*

- Supports Sender/Receiver Device authentication of Device OEM Server in a device-specific way
- Enables device to load/install Digital Key applet (if not done in factory)

2.5.9 *Vehicle OEM Server to KTS (5)*

- Provide key tracking data to KTS, such as
 - Public key of owner and receiver devices
 - Instance CA identifier of hosting SE (owner or receiver)
 - Anonymized vehicle identifier

2.5.10 *Sender/Receiver Device OEM Server to Vehicle OEM Server (6,8)*

- Establish trust relationship
- Exchange and sign the necessary certificates (see Section [16](#))
- Support Digital Key sharing
- Support Digital Key tracking
- Support Digital Key termination
- Support notifications.

2.5.11 *Fleet Management Server – Server-Based Owner Device Link (21)*

- FMS requests vehicle infleeting and defleeting from the SBOD.
- Prior to infleeting, the FMS proves ownership of the vehicle (using either standardized or proprietary mechanisms) to the SBOD.
- FMS requests the creation and termination of shared keys.
- FMS is notified by SBOD when owner Digital Key is paired with the vehicle. At this point, the vehicle is ready for receiver DK sharing.
- FMS sends the account binding information (between account identifier and device of the user which requested a key) to the SBOD.
- Request the SharingURL to be sent to a receiver device of the fleet management user.

2.5.12 *Vehicle OEM Server – Server-Based Owner Device Link (20)*

- Vehicle OEM server deploys the key material of the SBOD to the vehicle (in a proprietary way).
- Alternatively, vehicle OEM provides channel for the SBOD to conduct the infleeting transaction with the vehicle.
- Vehicle OEM server can provide proof of ownership for infleeted vehicles to the SBOD.
- Vehicle OEM server can verify the proof of ownership collected by the SBOD.

2.5.13 *Receiver Device OEM Server – Vehicle OEM Server Link (8) – Server-based Owner Device Server Link (20)*

- During the sharing process, the account binding information of the user device is sent via link (20) and link (8) to the SBOD, so that the SBOD can verify the same account binding information was received from the FMS.

2.5.14 *Server-Based Owner Device – Relay Server Link (16) – Receiver Device OEM Server Link (14)*

- This link enables cross platform key sharing as described in chapter 11 but with the SBOD performing the role of owner device.

2.5.15 *Receiver Device – Relay Server Link (14) – Server-Based Friend Device Link (19)*

- Eligible devices can share delegate keys to a SBFD.
- SBFD can share digital keys to devices to enable delegate services.

2.5.16 *Server-Based Friend Device – Vehicle OEM Server Link (17)*

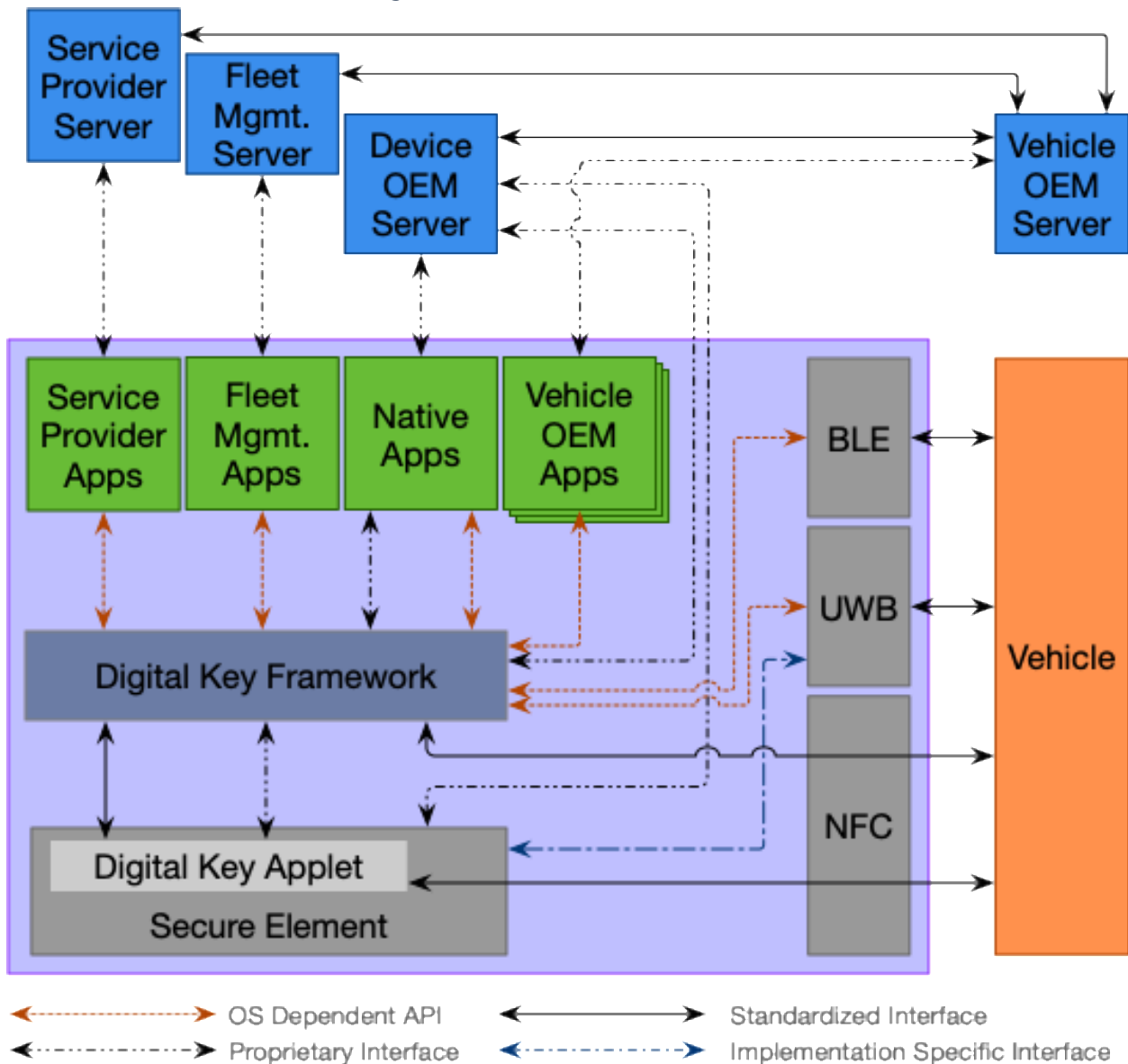
- Vehicle OEM cross-signs the public key of the Root CA of the SBFD
- SBFD manages its own shared keys.

2.5.17 *Receiver Device OEM Server – Vehicle OEM Server Link (8)*

- Receiver device may delete delegate keys and all keys shared by the delegate key.

2.6 Device Structure

Figure 2-5: Device Functional Elements.



On the device side, the following functional elements participate in the system: Secure Element and NFC controller are required components of eligible devices. The SE provides the root of trust, which is the starting point of the trust chain (see Section 16).

2.6.1 NFC Component [WCC1]

- Card emulation mode required for contactless transactions
- Host card emulation mode required for owner pairing

2.6.2 BLE Module [WCC2/WCC3]

- Communicate with the vehicle for owner pairing, First New Key Transaction and Digital Key transactions (lock/unlock, engine start, RKE etc.).
- Communicate with vehicle for setup of a secure ranging over UWB
- Communicate with vehicle for Remote transactions to allow the device to initiate on-demand features (e.g. Lock/Unlock etc.)
- Communicate with vehicle for transmission of Notifications to notify and signal change of state information
- Communicate with vehicle for transmission of data of 3rd party vehicle OEM application.

2.6.3 UWB Module [WCC3]

- Communicate with vehicle for secure ranging operation to determine the distance between device and vehicle in support of a secure distance measurement for passive entry and passive engine start functionality.

2.6.4 Secure Element (or equivalent)

- Java Card (example) and GlobalPlatform [20] support, SE Root based on GlobalPlatform-defined Controlling Authority Security Domain available
- GlobalPlatform Card Specification Amendment C [20] and GlobalPlatform Contactless Extension [21]
- Standard symmetric and asymmetric cryptographic support
- Hosts Digital Key applet
- Has the ability to distinguish communication between wired interface and contactless interface

2.6.5 Digital Key Applet

The Digital Key applet provides the following services:

- Hosts Digital Keys (one applet instance hosts Digital Keys of all Vehicle OEMs)
- Implements relevant (fast and standard) transactions
- Implements Instance CA (see Section 4.1 and Section 16.2.3) to support offline use cases and privacy protection
- Stores immobilizer tokens when required by the vehicle, offline attestations, access profiles, and other data associated with a Digital Key
- Verifies authenticity of the vehicle
- Verifies certificate chain of receiver public key

If the Digital Key applet is the SE-centric applet model as defined in Section 15.1, the applet also provides the following service:

- Verifies the Vehicle Public Key Certificate [K]

2.6.6 *Digital Key Framework*

- Implements the main features of owner pairing, Digital Key sharing, and management
- Provides common Digital Key functionality via a set of OS-specific APIs for Vehicle OEM apps. The APIs are described in documentation of the respective OS platform developer. The functional requirements are defined in [Appendix F](#).

2.6.7 *Vehicle OEM App*

- The Vehicle OEM app is optional. The main features of the app are supported natively by the device.
- May support the same features as the native app plus Vehicle OEM-specific features
- Provides ID&V with Vehicle OEM Server
- Retrieves owner pairing password
- Manages keys with non-standard access profiles

2.6.8 *Native App*

- Provides device-native UI such as Digital Key creation, Digital Key termination and deletion, Digital Key enable/disable, etc.
- Displays a list of all issued sender/receiver Digital Keys

2.7 **Vehicle States**

The vehicle has the following possible internal states:

- **Unpaired:** State in which no owner device is associated with the vehicle. This occurs either at first purchase or when the owner chooses in the vehicle menu to unpair, which leads to the deletion of all Digital Keys including the owner digital key.
- **Paired:** State in which an owner device is associated with the vehicle. This state is maintained when the owner deletes the associated owner device (e.g., to change the owner device) without deleting the receiver devices (see Section [13.5](#)).
- **Pairing:** The vehicle is in this state when waiting for an owner device to be associated.
- **Blocked:** The vehicle is in this state after a configurable number of failed pairing attempts (Vehicle OEM policy)
- **Private Vehicle:** A vehicle that has gone through owner pairing and has at least one key that originated through one or multiple P2P sharing steps from the owner key that still exists.
- **Fleet Vehicle:** A vehicle that has been in-fleeted and not yet de-fleeted. This can be a vehicle owned by a business entity or a private entity.
- **Un-Owned Vehicle:** Vehicle that is neither a private vehicle nor a fleet vehicle (e.g., during production, during transport, at dealership)
- **Infleeted:** State in which an SBOD is associated with the vehicle.
- **Infleeting:** The vehicle is in this state when waiting for an SBOD to be associated with it.

2.8 Digital Key User Roles

This section describes the different roles associated with Digital Keys. A single device can hold Digital Keys with different roles for multiple vehicles. Each role can only share Digital Keys for a subset of other roles, can view and delete all or only a subset of all Digital Keys for a vehicle. The role is defined through AccountRole (see [Table 2-3](#) and [Table 2-4](#)) in this Digital Key specification. View and deletion rights described for each role shall apply to all devices that have a Digital Key with this role for the vehicle. They shall also apply for the vehicle (either through the Vehicle UI or Vehicle OEM server) based on the AccountRole information provided in the Attestation Package (see [Table 11-13](#)).

2.8.1 Owner Role

The owner Digital Key is created by the vehicle via owner pairing or by the SBOD via infleeting. When an owner device is paired, the vehicle switches from the unpaired to paired state. The owner has all access rights to the vehicle. The owner's Digital Key may need to be registered in the KTS to be accepted by the vehicle.

The owner role can be shared only to the Digital Keys on devices within the same device OEM account as the owner device. All Digital Keys on devices in the owner account shall have owner role.

2.8.2 Shared Digital Key Roles

The vehicle accepts several shared Digital Keys. Shared Digital Keys may have restricted access rights to the vehicle. These access rights are assigned by the sender device using an Access Profile (see Section [Table 11-21](#)) when issuing the Digital Key and are checked by the vehicle and/or Vehicle OEM Server according to the Vehicle OEM policy. The shared Digital Key may need to be registered in the KTS to be accepted by the vehicle.

There are several roles of a shared Digital Key that are defined by visibility, manageability, and shareability settings (see [Table 2-2](#)). A Digital Key shared to different devices on the same device OEM account has the same role and is therefore named AccountRole. Visibility describes the ability of a device to show other existing Digital Keys to the user. Manageability describes the ability of a device to delete all or a subgroup of Digital Keys that it can view. Shareability describes the ability of a device to share Digital Keys with a specific AccountRole configuration. Visibility, manageability, and shareability of a shared Digital Key cannot be greater than of the Digital Key on the sender device.

2.8.3 General Rules for all Digital Key Roles

This section describes rules that apply to all Digital Keys. Digital Key Termination can occur (besides remote termination/management) as described below:

- via local deletion on the device or vehicle.
- mutually between all devices on the same device OEM account.
- via unpairing or unbinding of the vehicle.

Digital Keys on devices on the same device OEM account:

- can share Digital Keys of their own AccountRole to devices on the same device OEM account, if enabled by the vehicle OEM.
- can view Digital Keys on devices on the same device OEM account.
- must all have the same AccountRole.

Account Roles are defined in Section [2.8.4](#).

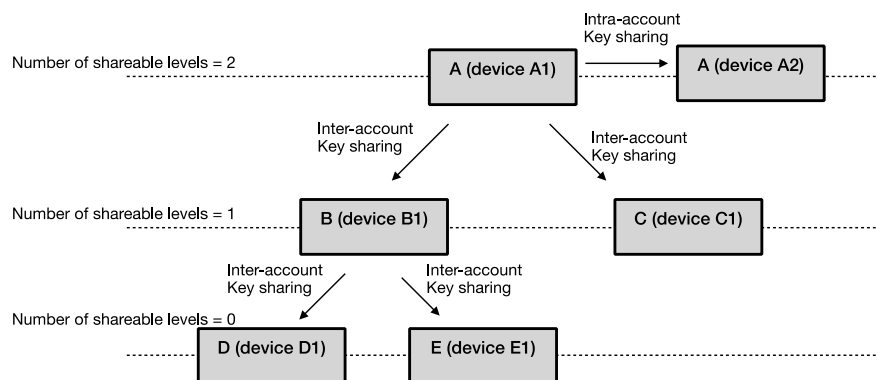
2.8.4 Account Role Definition

AccountRoles are defined based on their capabilities: visibility, manageability, and shareability.

Shareability can be set to unlimited or to a specific number of shareable levels (not Digital Keys). The number of shareable levels shall decrease by one at each new inter-account (receiver device and sender device belong to different device OEM account) Digital Key sharing. Intra-account (receiver device and sender device belong to the same device OEM account) Digital Key sharing can maintain the same visibility, manageability, and shareability setting (see [Section 2.8.3](#)) and the number of shareable levels shall remain the same upon each new intra-account sharing.

Figure 2-6 illustrates shareability through an example. A through E represent different device OEM accounts. Device A1 does intra-account sharing to device A2. The sender device A1 sets the number of shareable levels for the shared Digital Key (e.g., 2) and does inter-account Digital Key sharing to the receiver device B1 and C1. The number of shareable levels for the shared Digital Key is decreased by one, at the receiver device B1 and C1. The device B1 now acts as sender device and does inter-account Digital Key sharing with receiver device D1 and E1. Sender device B1 sets the number of shareable levels for the shared key to 1. The number of shareable levels is 0 at device D1 and E1. The device D1 and E1 cannot do inter-account Digital Key sharing for this Digital Key but can-do intra-account Digital Key sharing. Upstream Digital Keys of D1 and E1 are B1 and A1. Digital Keys of B1, C1, D1, and E1 are downstream shared Digital Keys of A1.

Figure 2-6: Illustration of number of sharing levels.



1

Table 2-2: Visibility, Manageability, and Shareability properties of AccountRole

Bit	B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
Meaning	Is Owner	Is Server	Is Static	RFU	Visibility				Manageability				Shareability			
					All	RFU	RFU	RFU	All *	RFU	RFU	Can share same manage-ability **	Intra-account	Number of shareable levels 111 = unlimited, 0xx = limited to xx		
Type 1	1	x	0	-	1	-	-	-	1+	-	-	1	x	1	1	1
Type 2	0	x	0	-	1	-	-	-	1	-	-	1	x	1	1	1
Type 3	0	x	0	-	1	-	-	-	1	-	-	0	x	1	1	1
Type 4	0	x	0	-	1	-	-	-	0	-	-	0	x	1	1	1
Type 5	0	x	0	-	1	-	-	-	0	-	-	0	x	0	x	x
Type 6	0	0	0	-	1	-	-	-	0	-	-	0	x	0	0	0
Type 7	0	x	0	-	0	-	-	-	0	-	-	0	x	1	1	1
Type 8	0	x	0	-	0	-	-	-	0	-	-	0	x	0	x	x
Type 9	0	0	0	-	0	-	-	-	0	-	-	0	x	0	0	0
Type 10	0	0	0	-	0	-	-	-	0	-	-	0	0	0	x	x
Type 11	0	0	0	-	0	-	-	-	0	-	-	0	0	0	0	0
Type 12	0	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-

Notes:

“*” means this Digital Key can delete all keys for the vehicle except the owner Digital Key.

“**” means this Digital Key can share same manageability setting indicated in bit B7. If B7 is set to 0, B4 has no effect but shall be set to 0.

“+” means the owner Digital Key can manage owner Digital Keys on other devices on the owner device OEM account.

“-” means the value (0 or 1) does not have an effect for this AccountRole.

“x” means values (0 or 1) are allowed and are not explicitly listed, not all combinations of multiple “x” values are possible/allowed/meaningful.

2

3 The [Table 2-2](#) shows how visibility, manageability, and shareability capabilities are set for
4 various accountRole values using 2 Bytes AccountRole field (see tag D5h [Table 11-20](#)).

5

6 Bit [B15] set to 1 indicates it is an Owner Digital Key. Otherwise, it is a non-owner Digital Key.

7 Bit [B14] set to 1 indicates it is a Server Digital Key. Otherwise, it is a non-server Digital Key.

Bit [B13] set to 1 indicates it is a static Digital Key such as plastic card or key fob. Otherwise, it is a non-static Digital Key. Visibility, manageability, and shareability do not apply to a Digital Key with bit [B13] set to 1, i.e., bits [B12, B0] are don't care and set to 0.

Bits [B8, B11] control visibility property of the Digital Key. Bit [B11] set to 1 allows the Digital Key to view all other upstream and downstream Digital Keys. Bit [B11] is set to 0 by default and allows the Digital Key to view all other downstream Digital Keys.

Intra-account Digital Keys are always to be able to view each other and are not controlled by bits [B8, B11]. Bits [B8, B10] are RFU. RFU bits shall be set to 0 by the sender and shall be ignored by the receiver in this Digital Key specification.

Bits [B4, B7] control manageability property of the Digital Key. Bit [B7] set to 1 allows the Digital Key to manage all other upstream Digital Keys with bit [B15] set to 0, and all other downstream Digital Keys. Digital Keys with bit [B15] set to 1 cannot be managed by Digital Keys with bit [B15] set to 0, even with bit [B7] is set to 1.

Bit [B7] is set to 0 by default and allows the Digital Key to manage all other downstream Digital Keys. Intra-account Digital Keys are always to be able to manage each other and are not controlled by bits [B4, B7].

Bit [B4] set to 1 allows the Digital Key to share same or lower manageability control to its downstream Digital Keys. It means that the Digital Key with bit [B7] set to 1 and bit [B4] set to 1, can share downstream Digital Keys with bit [B7] set to either 1 or 0. Bit [B4] set to 0 only allows the Digital Key to set bit [B7] equal to 0 to its downstream Digital Keys. If bit [B7] is set to 0 then bit [B4] shall be set to 0.

Bits [B5, B6] are RFU. RFU bits shall be set to 0 by the sender and shall be ignored by the receiver in this Digital Key specification.

Bits [B0, B3] control shareability property of the Digital Key. Bit [B3] set to 1 allows the Digital Key to do intra-account sharing. Otherwise, intra-account sharing is not allowed for the Digital Key. By default, B3 is set to 1 for the Digital Key. Intra-account sharing is unlimited and not controlled through bits [B0, B2].

Bits [B0, B2] control the number of shareable levels. Bits [B0, B1, B2] set to [1, 1, 1] then unlimited number of shareable levels are allowed for the Digital Key.

If bit [B2] is set to 1 then bits [B1, B0] must be set to 1. The following values for [B2, B1, B0] are disallowed: [1, 0, 0], [1, 0, 1], and [1, 1, 0]. If bit [B2] is set to 0 then bits [B0, B1] indicate the number of shareable levels allowed for the Digital Key.

The first owner Digital Key is created through owner pairing. This owner Digital Key can share Digital Keys with owner role to devices on the owner's device OEM account. The difference between the first owner Digital Key and the shared Digital Key(s) with owner role is that shared Digital Key(s) with owner role requires, as all shared Digital Keys, an Attestation Package signed by either the owner Digital Key or another shared Digital Key with owner role.

The types in [Table 2-2](#) and corresponding value of AccountRole field is shown in [Table 2-3](#). Intra-account Digital Key sharing is allowed in all types unless specified otherwise. A specific type can be hosted on a device or on a server. Both, intra-account Digital Key sharing capability and hosting environment are not changing the type of the key as per [Table 2-3](#), only the accountRole value changes. Note that not all server-types are explicitly specified.

Table 2-3: Type number and AccountRole value mapping.

Type number	AccountRole Value	AccountRole Description
Type 1	889F _h	Owner Digital key on all devices of the owner device OEM account
	C897 _h	SBOD, intra-account sharing not applicable (set to 0)
Type 2	089F _h	Digital key with same properties as an owner key, except it cannot delete an owner key
Type 3	088F _h	Digital key with same properties as type 2, except it can only share keys that can manage their downstream keys only
Type 4	080F _h	Digital key with downstream manageability but full visibility
Type 5	080B _h to 0809 _h	Digital key with same properties as type 4, except it can only share a limited number of levels
Type 6	0808 _h	Digital key with same properties as type 4, except it cannot share (except intra-account sharing)
Type 7	000F _h	Digital key with downstream visibility and manageability
Type 8	000B _h to 0009 _h	Digital key with same properties as type 7, except it can only share a limited number of levels
Type 9	0008 _h	Digital key with same properties as type 7, except it cannot share inter-account but can only do intra-account sharing
Type 10	0003 _h to 0001 _h	Digital Key with same properties as type 7, except it can only share a limited number of levels but not intra-account (Service use only)
	4003 _h to 4001 _h	SBFD key, intra-account sharing not meaningful
Type 11	0000 _h	Digital key with same properties as type 7, except it cannot share at all (no intra-account and inter-account sharing)
Type 12	2000 _h	Digital key on Plastic card

The DeviceType (tag 48_h in [Table 11-13](#)) identifies whether the hardware (plastic card, key fob, delegate server, phone, or watch) on which the shared Digital Key resides. Visibility for service keys on devices and delegate servers should be set to downstream-only for owner-paired vehicles for privacy protection.

[Table 2-4](#): shows the AccountRole types can be shared by each accountRole.

Table 2-4: Shareable types

Sender Type	Intra-account Receiver Type	Inter-account Receiver Type
Type 1	Type 1	Types 2 through 12
Type 2	Type 2	Types 2 through 12
Type 3	Type 3	Types 4 through 12
Type 4	Type 4	Types 4 through 12
Type 5	Type 5	Type 5 (decrease number of shareable levels by at least 1) Type 6 Type 8 (decrease number of shareable levels by at least 1)

Sender Type	Intra-account Receiver Type	Inter-account Receiver Type
		Types 9 and 12
Type 6	Type 6	None
Type 7	Type 7	Types 7 through 12
Type 8	Type 8	Type 8 (decrease number of shareable levels by at least 1) Types 9 and 12
Type 9	Type 9	None
Type 10	Type 10	Types 10 through 12
Type 11	none	none
Type 12	N/A	N/A

[Table 2-5](#): shows possible associations between AccountRole types and key entitlements (see [Table 11-21](#)).

Table 2-5: Entitlement association to AccountRole Types

Type	Device AccountRole Type Value	Server AccountRole Type Value	Entitlements
Type 1	0889F _h	C897 _h	Full
Type 2	089F _h	Not recommended	Entitlements with driving capability only
Type 3	088F _h	Not recommended	Entitlements with driving capability only
Type 4	080F _h	Not recommended	All (driving capability recommended)
Type 5	080B _h to 0809 _h	Not recommended	All (driving capability recommended)
Type 6	0808 _h	Not recommended	All (driving capability recommended)
Type 7	000F _h	Not recommended	All
Type 8	000B _h to 0009 _h	Not recommended	All
Type 9	0008 _h	Not recommended	All
Type 10	0003 _h to 0001 _h	4003 _h to 4001 _h	All
Type 11	0000 _h	Not recommended	All
Type 12	2000 _h	Not applicable	Not applicable

2.9 Access Profiles

For each device associated with the vehicle, the vehicle stores the Access Profile and corresponding public key.

The owner Digital Key has all access rights with no restrictions. As not all vehicles support all profiles, the vehicle indicates to the owner device which profiles are supported during owner pairing (see [Section 6](#)).

The sender may choose which profile to grant to the receiver during key sharing. It shall not be possible to grant more access and driving rights to the receiver relative to what the sender has. All Digital Keys with sharing capability should be able to share keys selected from a set of supported Access Profiles as defined in section 11. The vehicle OEM determines the available profiles via the “Entitlements” data structure in the uiBundle (see Table 17-46 json) as described in section 11 and thus determines the selection offered by the sender device.

The vehicle OEM server providing the uiBundle is responsible to not allow attribution of entitlements that increase the usage of vehicle compared to the sender device, e.g., sharing a “full” Digital Key from a “restricted” Digital Key. The vehicle OEM server shall verify this constraint at key tracking as well.

2.10 Versioning

2.10.1 General

Versioning is required so that devices, vehicles, and servers supporting different Digital Key specification releases, can still work together.

Version negotiation is used to agree on the highest version supported by all participants of a specific feature or (domain-) relationship.

Migration is introduced in this Digital Key specification and describes method to update current vehicle and device software (asynchronous to each other) to support SiaC feature defined in this Digital Key specification, without the need of a vehicle reset or the deletion of existing owner and shared Digital Keys.

The support for migration is highly recommended for the vehicle and the device. The migrationMetadata() API (see 17.7.1) allows devices to obtain new and updated information, such as device configuration, required to migrate Digital Keys to support SiaC feature. A Digital Key created using [41] can be converted to a Digital Key in Release 4 through Migration. However, Migration cannot change the hardware configuration, e.g., a WCC2 device or vehicle cannot become WCC3 through Migration

2.10.2 Domain Versions

Figure 2-7 identifies the domain versions that are relevant for the system.

Domain versions describe logical relationships, not physical APIs, between actors in the system. All APIs are associated to a particular domain version. Data structures might be associated with one or multiple domain versions, depending on how many entities receive and process the data structure. Data structures can contain elements that are determined by a different domain version than the data structure layout itself. This specification documents those associations where reasonable, so as to allow quick identification of the impact that an API or data structure change has on the relevant domain versions. Some data structures do not list domain version information. In case of a change in those data structures, the relevant domain version(s) need(s) to be identified depending on the nature of the change.

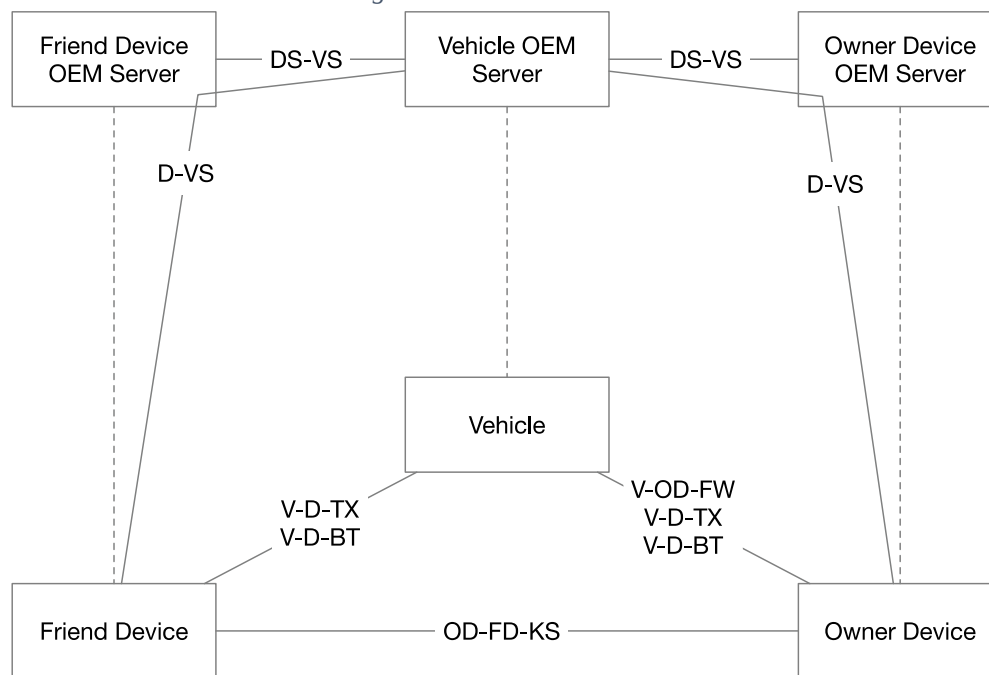
The domain version determining the data structure is provided in the first line (e.g., highest level TLV). Domain versions for data elements are provided with each element in the structure description.

Note that the APIs and data structures do not need to (but can) carry explicit version information, the knowledge of the API and data structure semantics and syntax are implicitly known by sender and receiver based on the associated domain version.

Note also that the relationships described by dashed lines in Figure 2-7 are out of scope of this specification and the representation of these relationships in any subsequent MSDs in this specification is for illustrative purposes only.

Domain versions always include a list of versions from both sides, for example, D-VS-serverList and D-VS-deviceList, or at least an agreed version from the responding side. When a commonly supported version is found through the exchange of lists, then this is the “agreed version”. Where relevant, this clarification is added throughout the specification.

Figure 2-7: Domain Versions



The vehicle-to- device (V-D) relationship is described by two domain versions.

1. (V-D-TX): Version of the transactions (fast, standard, etc.) between device and vehicle.
2. (V-D-BT): This domain version determines the Bluetooth (BT) version for Digital Key used between device and vehicle.

The vehicle-to-owner-device (V-OD) relationship is described by one additional domain version:

1. (V-OD-FW): Version of the framework relationship established at owner pairing and features based on this relationship.

The device-to-vehicle-server (D-VS) relationship and the device-server-to-vehicle-server relationship do not require a differentiation between owner (sender) and receiver device or owner (sender) and receiver device server. The versioning is managed independently of the key type.

The vehicle-side lists of V-D-TX, V-OD-FW and V-D-BT obtained by the owner are sent to the receiver device during key sharing. This allows Receiver device to determine if there will be an agreed V-D-TX and V-D-BT version to connect over Bluetooth LE and to transact with the vehicle. This allows to determine whether a shared key can be accepted or shall be rejected. The V-D-TX and V-D-BT finally used in transactions between the receiver device and vehicle may have different values than the V-D-TX and V-D-BT used in transactions between the owner device and vehicle. Owner and receiver device exchange sharing data to create shared keys. This data is partially generated and consumed by the framework and partially by the applet. Explicit applet domain version agreement is not required as the frameworks of both devices represent the applet and framework version in the key sharing domain version (OD-FD-KS). Vehicle OEMs provide endpoints URLs to device OEMs to enable routing of device information to the region and environment in which the vehicle is registered based on the ROUTING_INFORMATION obtained during owner pairing and key sharing by the devices. The vehicle OEM can define a D-VS and a DS-VS version per endpoint URL which allows vehicle servers to run on different SW releases per endpoint. See examples in Section 2.10.4

2.10.3 Domain Wise Version Agreement

2.10.3.1 V-OD-FW

The framework version is agreed between vehicle and owner device during owner pairing based on tags 5A_h and 5B_h in [Table 5-3](#) and [Table 5-4](#) respectively.

V-OD-FW-agreedVersion is provided during key tracking to the vehicle server (see Section [17.7.3](#))

If the device SW is updated to support a new V-OD-FW version, then the updated version list may be provided via the versionUpdate() API (see [17.8.3](#)) to the vehicle server. If the new agreed version requires Digital Key migration, the device calls migrationMetadata() API providing at least the new agreed V-OD-FW-agreedVersion.

If the vehicle SW is updated to support a new V-OD-FW version, then the updated version list may be provided via the versionUpdate API by the vehicle server to the device server. If the new agreed version requires Digital Key migration, then the device shall call migrationMetadata() API.

2.10.3.2 V-D-TX

The transaction version is agreed between vehicle and owner device during owner pairing based on tags 5C_h (both sides) in Table 5-4.

The vehicle list V-D-TX-vehicleList is provided during key sharing in Tag 5C_h in [Table 11-5](#). This allows the receiver device to verify version compatibility with the vehicle before accepting a shared key.

Independent of the V-D-TX version agreed during owner pairing or key acceptance, all devices agree on a version during every fast and standard transaction. This allows updating vehicle or device SW to support higher versions without the need to transfer this knowledge via servers to the other side. See tags 5C_h in [Table 15-31](#) AUTH0 Command Payload and [Table 15-12](#) SELECT Response

2.10.3.3 V-D-BT

The Bluetooth version is agreed between vehicle and owner device during owner pairing based on tags 5E_h and 5F_h in [Table 5-4](#) and [Table 5-5](#) respectively.

The vehicle list V-D-BT-vehicleList is provided during key sharing in tag 5E_h in [Table 5-4](#). This allows the receiver device to verify version compatibility with the vehicle before accepting or rejecting a shared key.

Independent of the V-D-BT version agreed during owner pairing or key acceptance, all devices agree on a version during every BT connection. This allows updating vehicle or device SW to support higher versions without the need to transfer this knowledge via servers to the other side. The negotiation of versions under the V-D-BT domain during owner pairing and passive entry is described in detail in Section 19.2.1 and Section 19.2.3

2.10.3.4 OD-FD-KS

The key sharing version is agreed between the sender device and the receiver device during key sharing based on tags 54_h and 55_h in [Table 11-5](#) and [Table 11-6](#).

The key creation request ([Table 11-5](#)) cannot be versioned and must be managed by adding new payloads with new tags if backwards-incompatible changes need to be done in the future.

Backwards-compatible changes can be done by adding new tags, as they shall be ignored by earlier versions. It is strongly recommended that devices plan for enough buffer space to receive larger key creation request messages than defined in this version of the specification.

2.10.3.5 D-VS

If the device OEM server receives the D-VS-serverList from the vehicle OEM server via trackKey() Response API (see [17.7.3.3](#)), then the D-VS-serverList is provided to the devices via the device OEM proprietary configuration method. In the absence of knowledge of D-VS-serverList, the device shall use the default value D-VS-serverList = 0100_h.

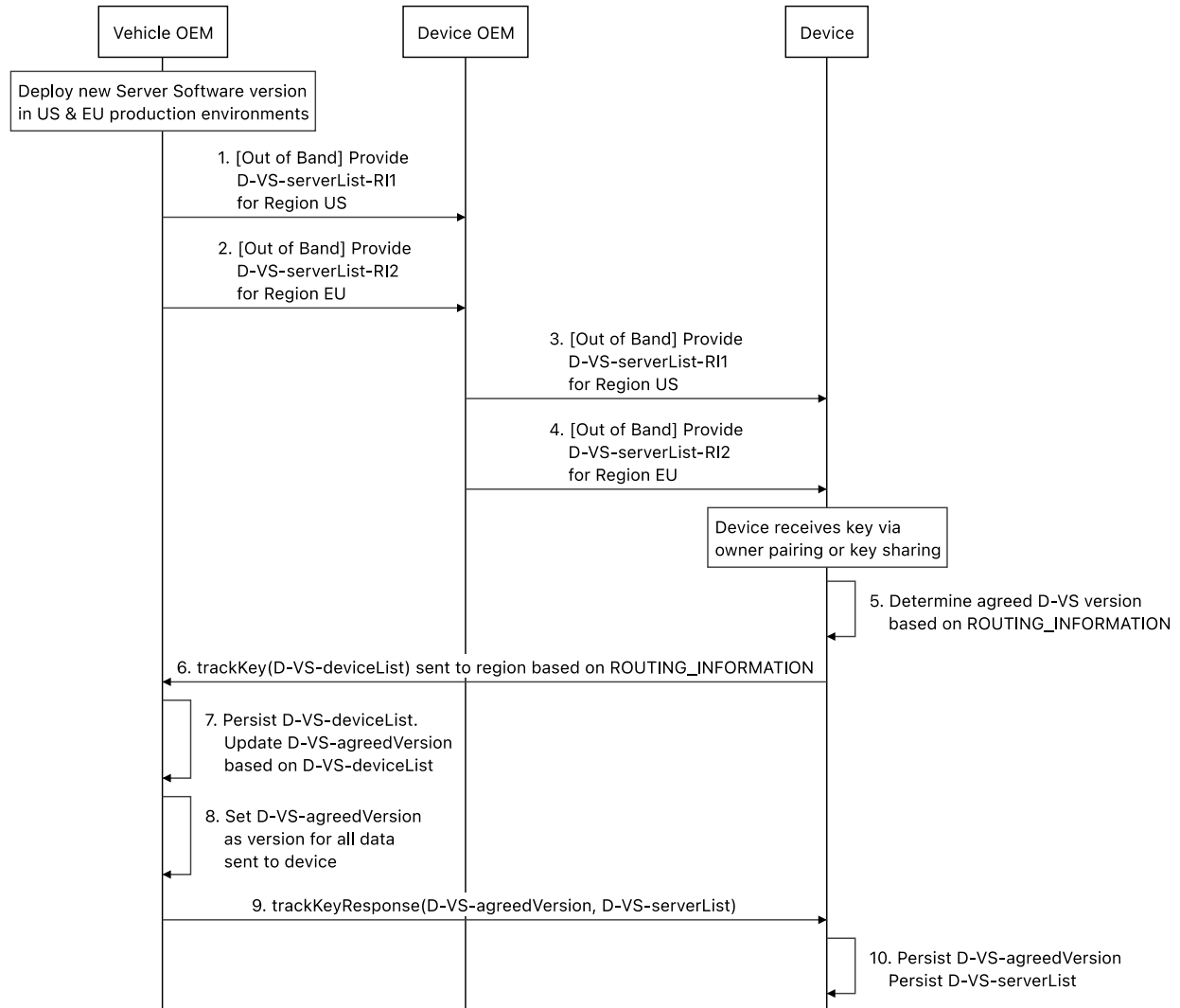
The device list D-VS-deviceList is provided during key tracking to the vehicle server (see [17.7.3](#)). In this Digital Key specification, trackKey() API as defined in (see [17.7.3](#)) shall be used. Device may send only the agreed version or all supported versions with the agreed version being the first, in the D-VS-deviceList.

The versionUpdate API is not required for this domain version.

One D-VS-serverList can be provided per endpoint URL by the Vehicle OEM to the Device OEM, which corresponds to a specific value of the ROUTING_INFORMATION (RI) provided by the vehicle at owner pairing and by the owner device at key sharing.

1

Figure 2-8: D-VS Agreement for Key Tracking¹



2

3

4

- 5 [1 – 2] VS deploys new SW version for some regions. When deployed, the VS provides
6 the new supported versions to connected DS in a proprietary way (based on the
7 working terms agreed between VS and DS) for each server endpoint identified by
8 the routing information (RI) as per above example (e.g., OEM1.USP.BRD1).
9 [3 – 4] DS provides the updated version lists in a proprietary way to all devices.

¹ Out-of-band in this figure refers to mechanisms that are outside the scope of the Digital Key Specification. E-mail or other mechanisms may be used for OEM-to-OEM communication compliant with pre-established agreements between device and vehicle OEMs; or Device OEMs may use proprietary APIs to communicate with devices similar to the telematics links deployed by Vehicle OEMs to communicate with Vehicles.

- [5] ROUTING_INFORMATION is used by the device to determine the agreed D-VS version to send the key tracking request based on e.g., D-VS-server-list-USP.
- [6 – 7] The key tracking request is sent by the device in the newly determined D-VS agreed version. The D-VS server list is determined using the routing information provided during owner pairing and the version information provided in Steps 1 – 4 to the device. Depending upon the vehicle server supported version, the key tracking request provides the device list for the D-VS version to allow the server to determine the agreed D-VS version for communication with this device (e.g., the key tracking response or a manage key call).
- Step 6: Although not shown in [Figure 2-8](#), the device also provides the list of supported V-OD-FW versions, including the agreed V-OD-FW version to the VS and may provide other supported versions as well.
- [8] From this moment on, (including the trackKey() Response message) the server sends all data in the new agreed D-VS version to this device. See [Figure 2-10](#) as an example.
- [9 – 10] The key tracking response contains the server D-VS version list and the agreed version, which is persisted in the device and should override the relevant configuration that the device has obtained in steps 3-- 4. It also contains the V-OD-FW-vehicleList to handle a device SW update that occurs after a vehicle SW update.

2.10.3.6 DS-VS

The device server to vehicle server version is agreed between device OEM and vehicle OEM via a proprietary configuration method. The versionUpdate API is not required for this domain version. It shall be possible to define different VS versions per region, environment, and brand.

Example:

OEM1.USP.BRD1:DS-VS = 0100_h, 0300_h, 0302_h

OEM1.EUPP.BRD1:DS-VS = 0100_h, 0300_h, 0302_h

OEM1.CNP.BRD1:DS-VS = 0100_h, 0300_h

2.10.4 Software Update Scenarios

A modification of the list of supported domain version values due to a SW update of one or more entities shall lead to a version agreement update between all entities impacted by the version change.

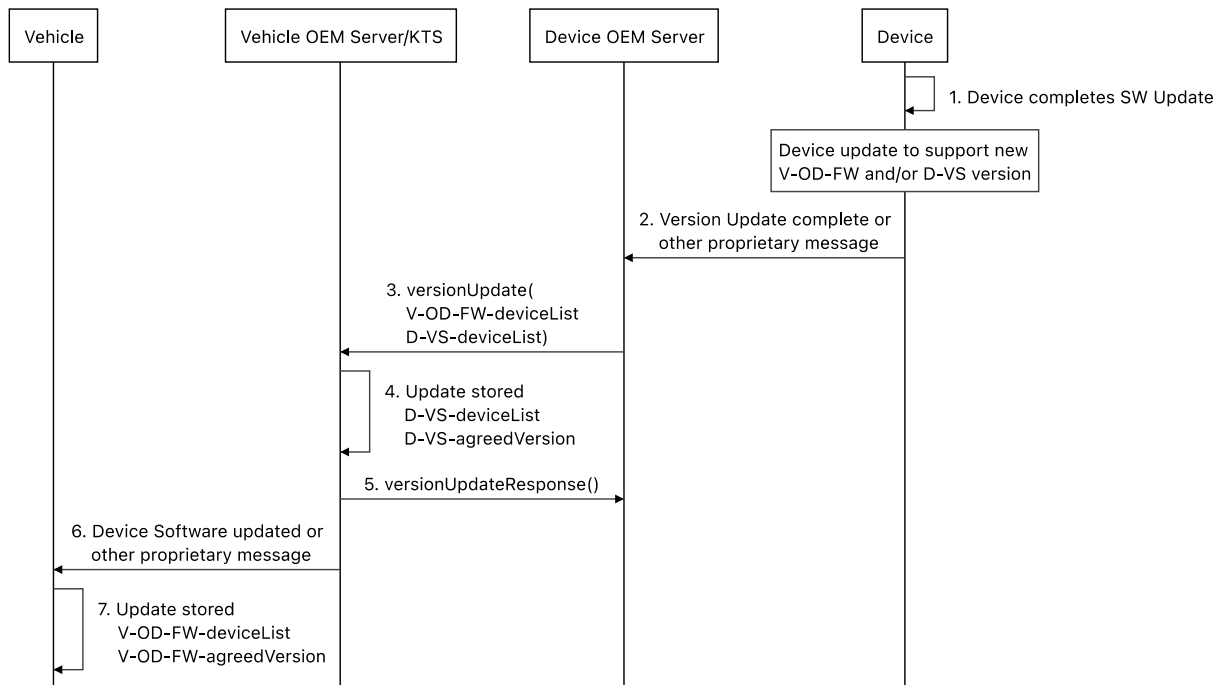
If required, each party needs to be able to handle temporarily different server versions on the other side on a per region basis. Therefore, the version information may need to be connected to a region-specific endpoint or server instance of the other OEM.

It is recommended that each OEM should always try rather to implement changes in a backwards compatible way, especially for server-side updates. Non-backwards compatible changes in server APIs should be avoided. The mechanism to inform OEM server or device counterparts is described in Sections 2.10.3.5 and 2.10.3.6.

2.10.4.1 Device Software Update

Software updates to the sender or receiver device that modify the list of supported V-OD-FW and/or D-VS versions may trigger the call of the versionUpdate() API to inform the vehicle and vehicle OEM server about the updated device version lists. The call to the vehicle itself might be asynchronous and/or the vehicle might be offline, so the response is not expected to contain vehicle side version information and since the vehicle side version has not changed in this case it does not need to be provided.

Figure 2-9: Version Agreement after Device Software Update



[1– 2] Device provides updated device version lists to device OEM server. The link between device and device OEM server is proprietary to the device OEM.

[3 – 5] Device OEM server calls versionUpdate API on vehicle OEM server, which determines new agreed version for D-VS. This could result in a new agreed version for V-OD-FW

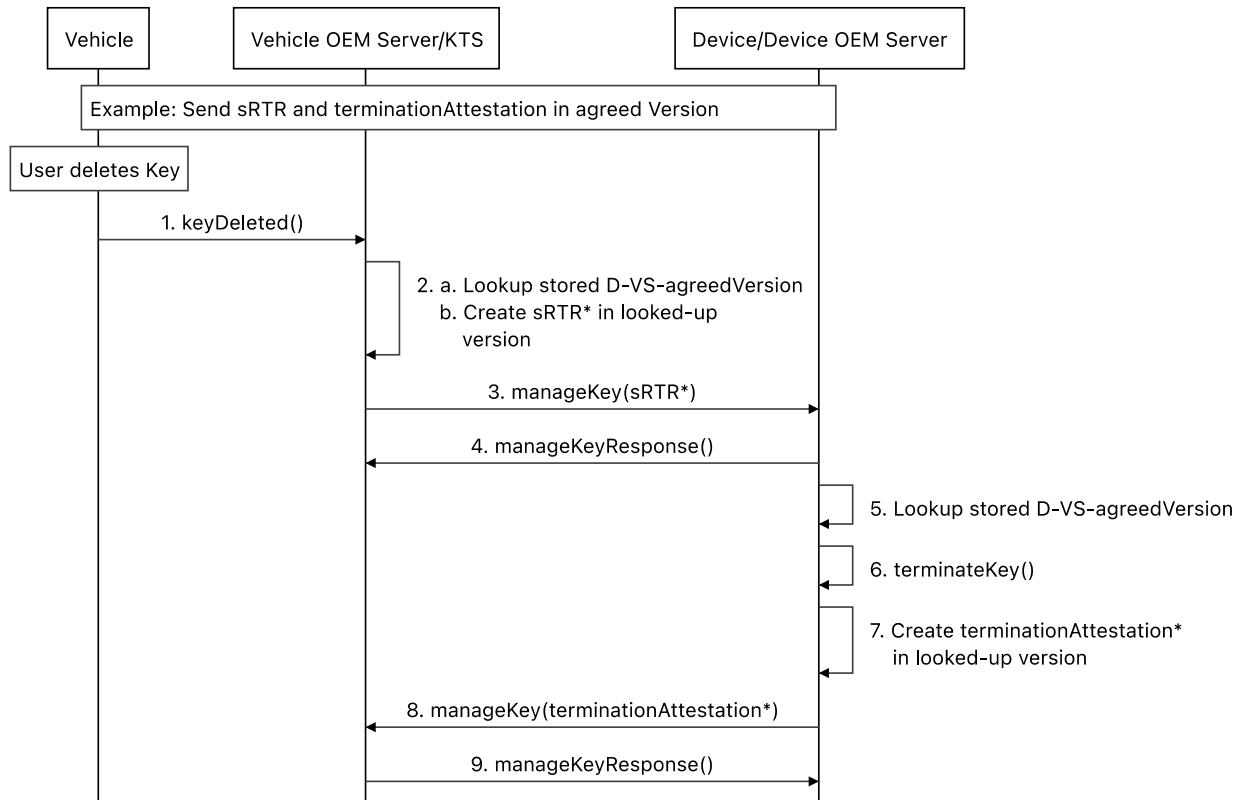
[6 – 7] Vehicle OEM server provides new V-OD-FW device list to vehicle which determines the new agreed version. The initiation of step [6] is asynchronous with and not contingent on the completion of step [5]. The link between vehicle and vehicle OEM server is proprietary to the vehicle OEM.

Alternatively, in the case that the vehicle has been updated before the device, the device may not call versionUpdate() when its software is updated, but either;

- a. use a previously received V-OD-FW-vehicleList to determine that migration is required, or
- b. wait until the vehicle software is updated and a versionUpdate() call is received. Then it follows flow as depicted in [Figure 2-11](#).

1

Figure 2-10: Example – manageKey() usage based on D-VS agreed version



2

3 Data elements exchanged between the server and the device, such as the server remote
4 termination request (sRTR, see Section 14.2) and terminationAttestation are versioned based on
5 the D-VS versions as shown in the example in [Figure 2-10](#). In the example above, sRTR* and
6 terminationAttestation* are versions of the sRTR and the terminationAttestation that are
7 accepted by the device and the server in the agreed D-VS version.

8 Relationships between device and device OEM server as well as vehicle and vehicle OEM server
9 are proprietary, and their version management is out of scope.

10 2.10.4.2 Device Server SW Update

11 In case a device OEM server SW update that modifies DS-VS-serverList the device OEM may
12 notify the vehicle OEM using a proprietary mechanism that is deemed acceptable by both,
13 vehicle and device OEM. The endpoint of the device OEM server that received the update shall
14 increment or rollback the version that is included in the URL, depending on the applicable
15 specific region or instance of the vehicle OEM server and exact API call
16 (e.g., .../v3/manageKey). The details of the URL scheme are provided in [17.5](#).

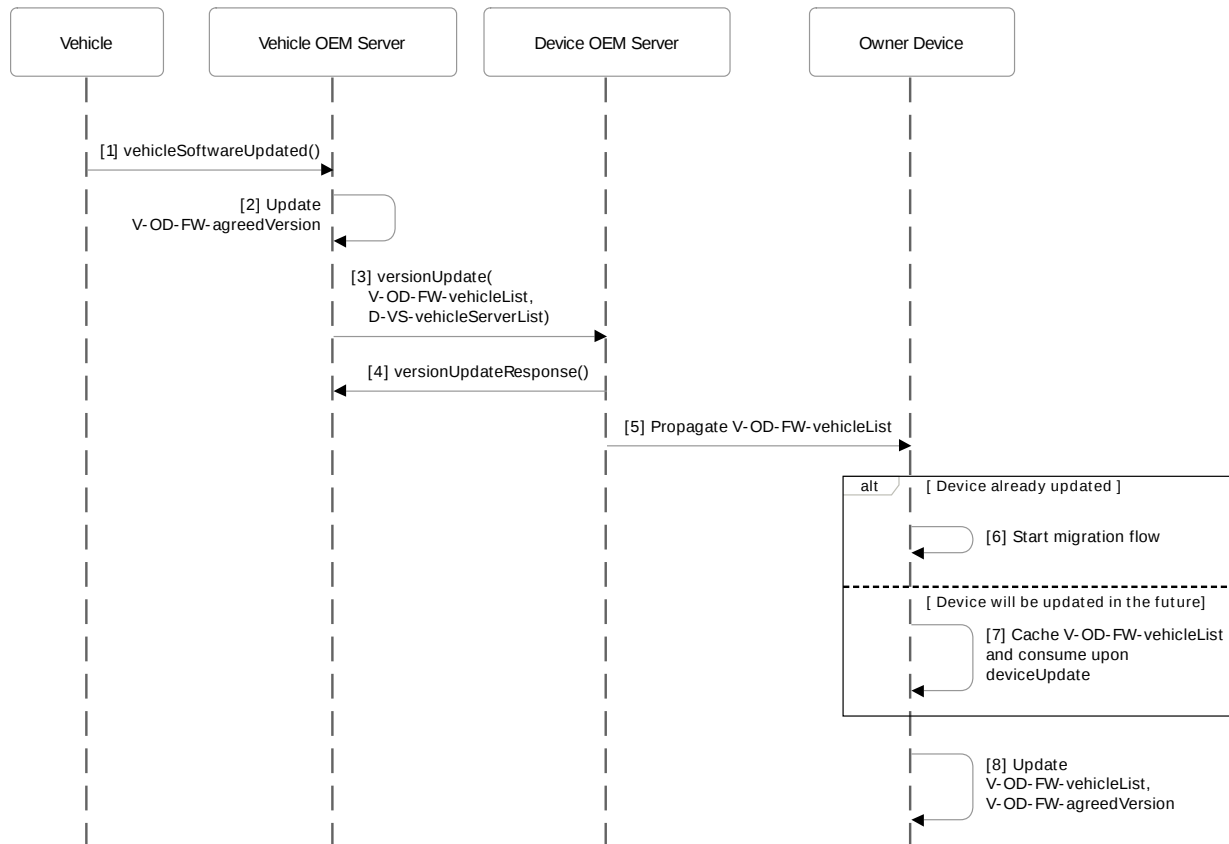
17

18 2.10.4.3 Vehicle SW Update

19 Software updates of the vehicle that modify the list of supported V-OD-FW shall trigger a call of
20 the versionUpdate() API to inform the owner device via the device OEM server about the new
21 vehicle version list (V-OD-FW-vehicleList).

1 Note: Updates that impact V-D-TX and V-D-BT are agreed during a Fast Transaction or
2 Standard Transaction with the vehicle.

3 *Figure 2-11: V-OD-FW Agreement after Vehicle SW Update*



- 4
- 5 [1] Vehicle SW update is confirmed to the server.
- 6 [2] Vehicle server updates agreed V-OD-FW version.
- 7 [3 – 4] Vehicle SW versions are sent to device server
- 8 [5] Device server provides new vehicle SW version lists to device using a proprietary
- 9 message.
- 10 [6] Start migration flow if device is already updated, else device stores the new V-
- 11 OD-FW-vehicleList and consumes it when device SW is updated
- 12

13 Note that an update to a server version that impacts D-VS-serverList is configured in the devices

14 using a proprietary method, as described in Section 2.10.4.4.

15 2.10.4.4 Vehicle Server SW Update

16 In case a vehicle OEM server SW update that modifies DS-VS, the vehicle OEM notifies the

17 device OEM using a proprietary mechanism that is deemed acceptable by both device and

18 vehicle OEMs. The endpoint of the vehicle OEM server that received the update shall increment

19 or rollback the version that is included in the URL. This step is dependent on the region-specific

server instance, if any, and exact API call (e.g., .../v2/trackKey). The details of the URL scheme are provided in Section [17.5](#).

In case of a vehicle OEM server SW update that modifies the D-VS-serverList, the vehicle OEM notifies the device OEM in a proprietary way. The device OEM shall propagate this information to relevant devices using proprietary mechanisms (as shown in [Figure 2-8](#)), so that devices are made aware in advance of changes to the D-VS-serverList per region, environment, and brand for every vehicle OEM.

The D-VS-serverList is provided in a proprietary way to every device OEM so that devices are already aware of the D-VS-serverList per region, environment, and brand for every vehicle OEM.

Example:

OEM1.USP.BRD1: D-VS-serverList = v1.0, v2.0, v2.1; DS-VS = v1.0, v2.0, v2.2

OEM1.EUP.BRD1: D-VS-serverList = v1.0, v2.0, v2.1; DS-VS = v1.0, v2.0, v2.2

OEM1.CNP.BRD1: D-VS-serverList = v1.0, v2.0; DS-VS = v1.0, v2.0

Devices that have existing keys for this vehicle OEM re-calculate the agreed D-VS version for each key immediately, whereby the re-calculation is based on the routing information obtained for the corresponding vehicle (either at owner pairing or at sharing).

Device to vehicle server communication is then based on the new agreed version. A new key would send the key tracking request based on the updated D-VS agreed version.

2.10.5 Migration Scenarios

All devices having a Digital Key for the vehicle as well as the vehicle itself get SW updated at different moments in time. Every SW update triggers a new version agreement that can lead to execution of migration, in which case the system functionality is partially or fully moved to the new functionality described in this Digital Key specification.

Migration flows consist of the following general steps. Note that not all steps need to be executed in all situations.

1. Device or vehicle server call versionUpdate() to announce a change in the supported D-VS version and/or V-OD-FW version.
2. Device server calls migrationMetadata() to obtain situation-dependent migration data for the device. If the migration fails at the vehicle server, then vehicle server responds with appropriate sub-status error code. For example, if migration fails due to invalid migration version, then the vehicle server shall respond with 50128 sub-status error code (see [Table 17-68](#)).
3. Device server calls trackKey() to commit a successful migration process. If key conversion (see section [2.10.11](#)) during migration on the device fails then trackKey() shall not be called by the device server. If trackKey fails on the vehicle server during migration, then vehicle server responds with appropriate sub-status error code (e.g., 50130 sub-status error code (see [Table 17-68](#))).

2.10.5.1 No Migration

No migration occurs when the vehicle SW is updated to support SiaC feature defined in this Digital Key specification, but the sender and receiver devices are not yet SW updated to support SiaC feature.

Upon vehicle SW update, the vehicle server calls versionUpdate() to indicate the new V-OD-FW-vehicleList to the device server and the devices. The vehicle server calls versionUpdate() for each key (ID).

2.10.5.2 Partial Migration

Partial migration occurs when the device is SW updated to support SiaC feature defined in this Digital Key specification but not the vehicle, or if vehicle server SW but not the vehicle SW is updated to supported SiaC feature while the devices SW already support SiaC feature. In other words, partial migration is triggered by the D-VS agreed version = 0300_h and vehicle server provides new uiBundle and sharedAccountsData (see [Table 17-41](#)) to the device. Although partial migration does not depend on the vehicle, it is recommended to follow the same “first owner, then friends” hierarchy as for full migration. Device Digital Key account roles for partial migration are defined as follows:

- Owner device is assigned the owner role, Vehicle OEM Server shall only indicate friend role (type 11) for sharing (in uiBundle/sharingInfo/supportedEntitlements)
- All friend devices, including owner watch, are assigned the friend role (type 11)

See [Table 2-3](#) for the full list of account roles and types.

The purpose of partial migration is to migrate to the usage of V3 server APIs and new uiBundle and new format of sharedAccountsData.

If a vehicle server SW is updated to support SiaC feature while the devices SW already support SiaC feature, a bulk partial migration on many devices can occur. The timing of a bulk partial migration should be managed as smoothly as possible without causing overload situations but is nonetheless out of scope of this Digital Key specification.

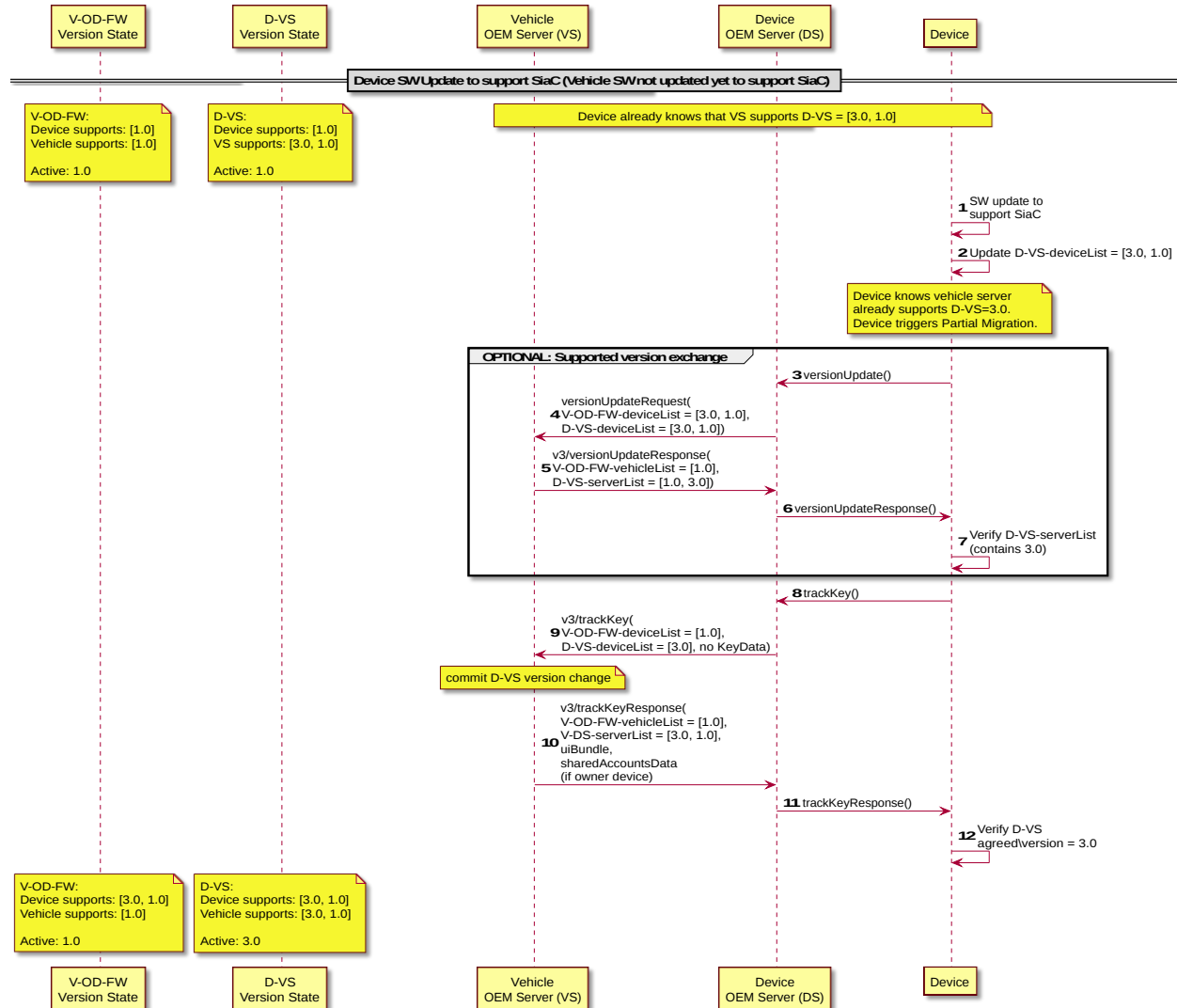
[Figure 2-12](#) describes partial migration flow after the device SW is updated to support SiaC. Both device server and vehicle server already support DS-VS version v3 API calls (see section 17). Upon device SW update, the D-VS-deviceList = [0300_h, 0100_h] is updated. The device already knows that vehicle server supports D-VS 0300_h and therefore triggers partial migration. The device may call v3/versionUpdate() to provide the updated D-VS-deviceList to the vehicle server or the device may call trackKey() with the purpose to obtain the new uiBundle (see [Table 17-46](#)) and migrate to the new UI structure. This key tracking request (see [Table 17-5](#)) does not contain encryptedKeyData for owner device([Table 6-3](#)) and for friend device² ([Table 11-17](#)). Therefore, for friend keys only, the vehicle server only returns new uiBundle but no encryptedDeviceData (see [Table 17-6](#)) in key tracking response.

If the updated device is an owner device, then new sharedAccountsData (contained in encryptedDeviceData) is also returned in new format by vehicle server. The trackKey() request shall contain D-VS-deviceList = [0300_h] and may contain further supported device versions (see [Table 17-5](#)). The trackKey() request has accountIdHash, which is defined in [\[41\]](#). During partial migration only, the vehicle OEM server shall generate and assign the groupIdIdentifier using the

² The term friend device is used here since the device has not yet been upgraded to a version that supports Sharing in a Chain

accountIdHash instead of accountInfoHash. The process of creation of the groupIdIdentifier is described in Section 11.4.4.

Figure 2-12: Partial Migration after Device SW Update to support SiaC.



2.10.5.3 Full Migration

Full migration occurs first on the owner device if the vehicle and owner device are both SW updated to support SiaC feature. After the migration of the owner Digital Key, the owner Digital Key can share keys with all new accountRoles (see section 2.8.4). Friend Digital Keys are converted during device update if the vehicle and owner device are already SW updated. Full migration may be triggered by the device SW update to support SiaC feature or vehicle SW update to support SiaC feature, while all other participants are already updated to support SiaC feature. A partial migration may have occurred before a full migration is triggered, depending on the SW update order of the vehicle and devices. The purpose of full migration is to convert the

relevant endpoint structure in the applet to support re-sharing (sharing in a chain) with the goal to migrate existing Digital Keys to their new roles in the framework and UI (see [Table 2-3](#)).

The following use cases can be distinguished:

1. Migration of owner-paired device
2. Migration of device OEM accounts that already have a device with V-OD-FW = 0300_h (V3 key) with an account role type that allows sharing
3. Migration of device OEM accounts that only have V-OD-FW = 0100_h (V1 friend keys)

Use case 1 comprises the migration of the owner-paired device to the V3 owner role (type 1).

Use case 2 comprises the V3 owner-paired device (either owner-paired in V1 and then migrated to V3 or owner-paired in V3) as well as V3 shared keys (e.g., type 2) with both having V1 friend keys on the same device OEM account. In this case the target account role for migration of V1 keys shall be the role that is already assigned to the V3 key on the same device OEM account.

Use case 3 comprises all device OEM accounts that only have devices with V1 keys.

In use case 1 there is no need to indicate the target account role in the owner re-tracking request ([Table 2-6](#)).

For use cases 2 and 3, the device OEM and vehicle OEM shall agree upon a policy which defines valid target account roles for key migration. The establishment of this policy is out of scope of this specification. During migration, the device may indicate a target account role (Tag D5_h) compliant with this migration policy in the friend re-tracking request ([Table 2-7](#)). If indicated by the device, the vehicle OEM server shall verify compliance of the received target account role with the migration policy and accordingly accept or reject the re-tracking request. If the device does not indicate the target account role, then the vehicle OEM server shall determine the target account role based on the agreed migration policy.

[Figure 2-13](#): illustrates full migration flow when triggered by the vehicle. The vehicle SW update is indicated by calling v3/versionUpdate() including V-OD-FW-vehicleList = [0100_h, 0300_h]. The device determines V-OD-FW agreed version = [0300_h] is possible, and calls the v3/migrationMetadata() to obtain new endpoint creation data, device configuration, and mailbox mapping. Then the device converts the endpoint (see [2.10.11](#)), enables sharing and then tracks the new endpoint certificate by calling v3/trackKey with only the endpoint certificate (see [Table 2-6](#); and [Table 2-7](#):). The key tracking response provides the new roles and entitlements in the uiBundle (see [Table 17-46](#) json of uiBundle). During full migration, the vehicle OEM server shall be responsible for generating and assigning a unique groupIdIdentifier using the accountInfoHash, which is defined in section [6.3.4.3](#). During full migration, the vehicle OEM server shall not calculate groupIdIdentifier using the accountIdHash. The determination of the groupIdIdentifier using accountInfoHash is described in Section [11.4.4](#).

Table 2-6: Owner Key Tracking Request for Re-Tracking in Full Migration

Tag	Length (Bytes)	Description	Field is	Domain Version
7F45 _h	variable	Owner key re-tracking request		V-OD-FW
	[H] Digital Key certificate as per Figure 6-7 and Table 11-9		mandatory	V-OD-FW

	5E _h	32	Account Info Hash	mandatory	D-VS
	5F49 _h	65	Device privacy encryption key (Device.Enc.PK)	mandatory	D-VS
	DA _h	Variable (maxlen: 10 bytes)	Device privacy encryption version Default: “ECIES_v1” (ASCH)	mandatory	D-VS

1

2

Table 2-7: Friend Key Tracking for Re-Tracking in Full Migration.

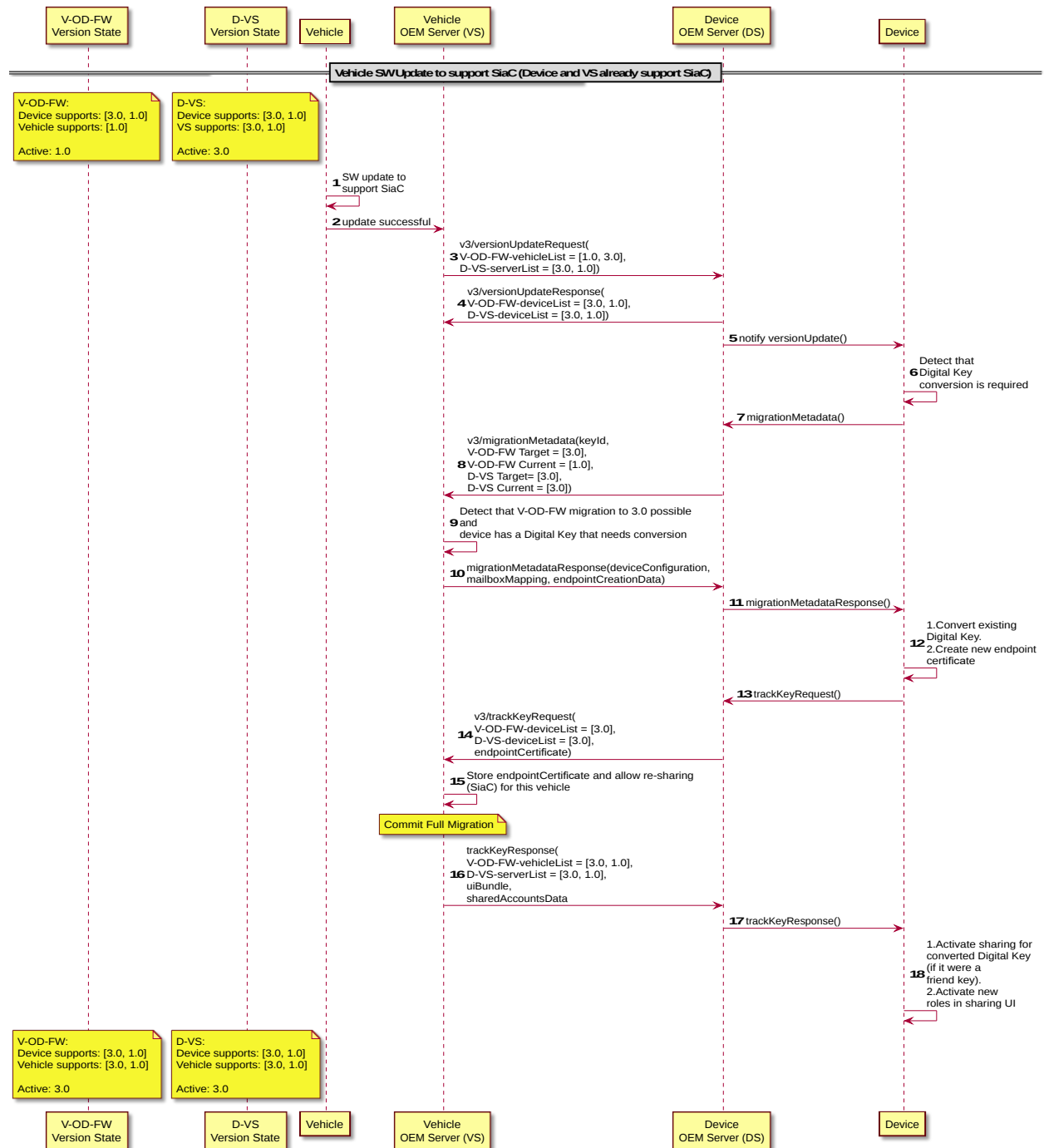
Tag		Length (Bytes)	Description	Field is	Domain Version
7F48 _h		variable	Friend key tracking and online attestation delivery request		V-OD-FW
	Friend endpoint certificate as per Table 11-9 signed by Instance CA			mandatory	V-OD-FW
	Friend instance CA certificate as per Table 11-8 signed by external CA private key			mandatory	V-OD-FW
	D5 _h	variable	Target accountRole (see Table 2-2:)	optional	V-OD-FW
	5E _h	32	Account Info Hash	mandatory	D-VS
	5F49 _h	65	Device privacy encryption key (Device.Enc.PK)	mandatory	D-VS
	DA _h	Variable (maxlen: 10 bytes)	Device privacy encryption version Default: “ECIES_v1” (ASCH)	mandatory	D-VS

3

4 accountIdHash in trackKey() (see [Table 17-5](#)) is employed by the vehicle OEM server to manage
5 both older versions of Digital Keys and Digital Keys created using this Digital Key specification.
6 The accountIdHash allows the vehicle OEM server to associate these Digital Keys with the same
7 Device OEM account identifier, ensuring proper tracking and management across different
8 versions. Note that [Table 2-7](#) refers to “friend keys” as migration object for clarity. When
9 migration is completed, the key becomes a “shared key” as the term “friend” does not exist in
10 this specification, except in reference to [\[41\]](#).

1

Figure 2-13: Full Migration after vehicle SW update.



2

- 3 [1 – 2] Vehicle SW update is confirmed to the vehicle server.
- 4 [3] Vehicle server calls /v3/versionUpdate() to inform device server about version change. Updated version information is included.
- 5
- 6 [4 – 5] Device is notified and detects that conversion is required.
- 7 [6 – 11] Device requests migration data from vehicle server using migrationMetadata().
- 8 Vehicle server provides all information to the device that is required to share

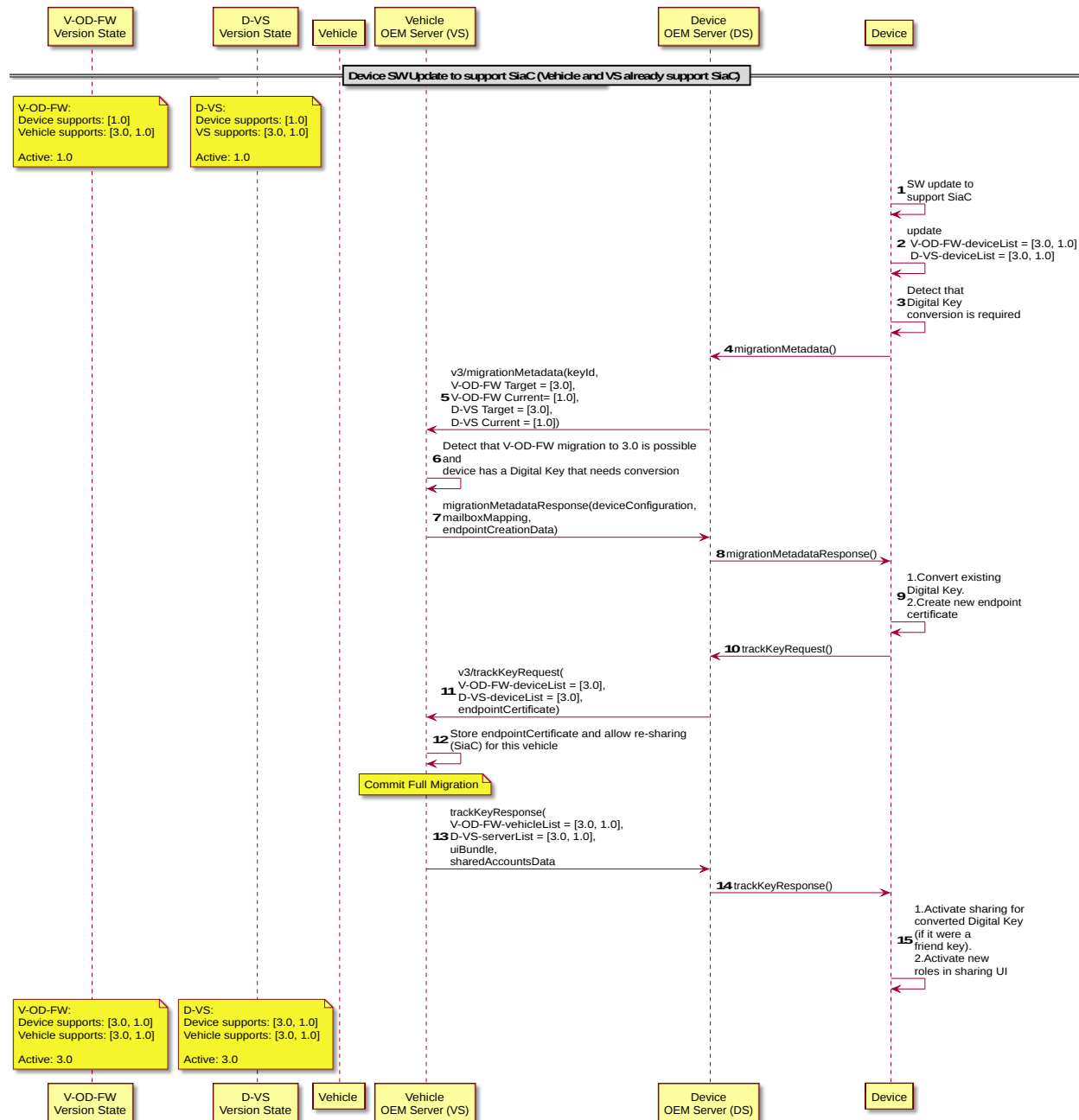
- 1 Digital Keys (endpoint creation data, device configuration, and mailbox
2 configuration).
- 3 [12] Device converts keys (See [2.10.11](#))
- 4 [13 – 17] Device tracks the converted endpoint certificate using /v3/trackKey(). Vehicle
5 server responds with new accountRole and entitlements that allow the device to
6 share keys on its new converted role. The encrypted key tracking data is defined
7 in [Table 2-6](#) (owner) and [Table 2-7](#) (friend).
- 8 [18] Device updates UI to show shared Digital Keys and allow key sharing as per new
9 role of the Digital Key.

10

11 [Figure 2-14](#) describes full migration flow triggered by device SW update to support SiaC feature.
12 Before the device SW is updated, the device server is already aware that the vehicle and vehicle
13 server SW is updated to support SiaC feature. The device determines new V-OD-FW agreed
14 version and D-VS version 3.0, is possible. The new V-OD-FW agreed version = [0300_h],
15 requires the device to migrate the existing Digital Keys for the vehicle. Therefore, the device
16 calls the /v3/migrationMetadata() on the vehicle server to obtain the new endpoint creation data,
17 device configuration data, and mailbox mapping. It then converts the endpoint, enables sharing,
18 and then tracks the new endpoint certificate by calling v3/trackKey with only the endpoint
19 certificate (see [Table 2-6](#): and [Table 2-7](#):). The key tracking response provides the new roles and
20 entitlements in the uiBundle (see [Table 17-46](#) json of uiBundle).

1

Figure 2-14: Full Migration after device SW update.



2
3

2.10.5.4 Migration Control

The vehicle server controls the migration sequence by calling versionUpdate() on the owner device server if the migration is triggered by the vehicle SW update (see Figure 2-13:). The vehicle server shall call versionUpdate() on all receiver devices only after full migration of the owner device is successful. The receiver devices cannot migrate until the owner device successfully migrates, otherwise the new accountRoles (see section 2.8.4) and multiple upstream/downstream devices with shared keys cannot be represented on the owner device UI.

Device determines the target role to be included in re-track request as per migration policy described in section [2.10.5.3](#). The enforcement of the migration policy is performed by the Vehicle OEM server.

If the owner device migration did not occur after the versionUpdate() call, likely because the owner device SW is not yet updated to support SiaC feature, then vehicle server shall not call versionUpdate() on receiver device servers until the owner device SW update has triggered a successful full migration (see [Figure 2-14](#)).

If the receiver device calls migrationMetadata() but the owner device is not migrated yet, then the vehicle server shall reject this migrationMetadata() call with 50129 sub-status error code ([Table 17-68](#)). Note that this should not happen because the receiver device is not informed by the vehicle server that the vehicle SW is updated until the owner device migration is successful.

If the migration fails, the vehicle server shall not migrate to a different domain version (V-OD-FW, D-VS) than the currently agreed version. All entities shall continue to operate on the domain version that is active before the start of migration. The device should retry migration until it is successful or maximum failed migration attempts limit is reached. The resolution of failed migration is out of scope of this Digital Key specification.

2.10.6 Version Numbers

Versions consist of a major and a minor version number (e.g., v2.3 with 2 = major and 3 = minor).

Major versions are used for non-compatible versions of a data structure or API, or to add or remove features, data structures and APIs.

Minor versions (with same major version) are compatible but limit the available feature set to the one defined by the lower minor version.

To achieve forward compatibility, data elements in structures/commands/APDUs/API parameters that have been added in higher versions should be ignored by the entity with the lower version for TLV and JSON structures. (e.g., the key creation request data structure (Tag 7F31_h)).

Entities shall agree on an exact version that is mutually supported. Communication based on different versions used by each entity (e.g., vehicle supporting versions 2.1 and 2.2, device supporting 1.0, 2.0 and 2.3) shall not occur.

Introduction of anchor versions (in section 2.10.7) allow both entities to find a commonly supported version.

2.10.7 Version Introduction

With the introduction of this Digital Key specification impacted domain versions shall be set to 0300_h as stated in [Table 2-9](#).

It is not required to implement Digital Key specification version 1.2.x before implementing this Digital Key specification.

The following plan describes the smooth introduction of versioning into the system without breaking compatibility with existing deployments of vehicles, servers and devices.

- All commands, responses, tags or data elements shall be transmitted as described in Digital Key Specification v1.0.0 or v1.1.0 between devices and vehicles if one entity only

- supports Domain Version 1.0 of V-OD-FW (referred to as SPAKE2+ Protocol version 1.0 in Digital Key Specification v1.0.0 or v1.1.0).
- For all other cases, any attempts to negotiate versions by an entity as defined in this specification shall be gracefully ignored by the other entity (i.e., the commands and responses shall be processed as if the unknown version elements are absent).
 - All senders should start implementing and using the new data structures and API versions for each relevant domain version 3.0.
 - All receivers that gracefully ignore attempts to negotiate versions are considered to support the CCC Digital Key Release 3 specifications v1.0.0 and v1.1.0.
 - Therefore, in the event that a versioning capable device or vehicle determines that it is interoperating with a non-versioning capable vehicle or device, the sender shall revert to using features defined in CCC Digital Key Release 3 specifications v1.0.0 and v1.1.0.

2.10.8 Version Support

Not all entities may be able to support all versions immediately. Therefore, anchor versions that must be supported by all entities, as a coordinated fallback version, shall be used in order to ensure interoperability. Anchor versions are not dedicated parameters, they are regular version values in the domain version lists, which are determined to be mandatory for all entities. An entity shall support all anchor versions prior to its currently supported version. E.g., if the device supports domain version 8.2, it shall support all anchor versions less than and equal to 8.2.

Backward compatibility to older versions is intended for fallback purposes only. After Owner Pairing, the vehicle only supports the current owner pairing version for new pairings until all keys are deleted. Then, the vehicle will support again all versions.

Anchor versions shall be agreed upon in the CCC Technical Working Group and updated in [Table 2-9](#).

The definition of a grace period to support new versions in all entities after a CCC specification release is outside of this specification. Deprecation of specific data structures/versions/APIs shall be documented in this specification.

Support period and deprecation rules for older versions are defined outside of this specification.

2.10.8.1 Maintaining compatibility across Versions

To maintain compatibility across differing domain versions supported by devices and vehicles the following rules shall apply:

1. To prevent buffer overflows on the recipient, while also allowing for future incremental additions as needed, a transmitter shall not exceed the maximum size of the SELECT, SELECT Response, SPAKE2+ Request and SPAKE2+ Verify Commands as defined in [Section 5.1](#) by 128 bytes.
2. Using the messaging schemes defined in this specification, a CCC Digital Key Product shall identify the version of the product it is interoperating with. Upon identification of the domain version supported by the DK message recipient, the DK message transmitter shall not transmit a message that includes a tag or data element that has not been defined

- in the specification version supported by the recipient. The domain versions and associated specification versions within which they are defined are outlined in [Table 2-9](#).
3. Servers shall ignore the reception of the data elements as outlined in [Table 2-8](#), even if they don't implement versioning as defined in CCC Digital Key Specification v1.2.0.

Table 2-8: Reception of Versioning Parameters by non-versioning capable Servers

Section	API	Recipient	Parameters (JSON)	Notes
17.7.3.2	trackKey	Vehicle OEM Server	vehicleOwnerDeviceFramework-deviceList (vodfwDeviceList) deviceVehicleServer-DeviceList (dvsDeviceList)	
17.7.3.3	trackKey() Response	Device OEM Server	serverSupportedVersions (dvServerList)	If trackKey message does not include vodfwDeviceList or dvsDeviceList
17.8.3.2	versionUpdate	Vehicle OEM Server Device OEM Server		Device OEM calls/tries to call versionUpdate() API on vehicle OEM server Vehicle OEM calls/tries to call versionUpdate() API on Device OEM server

2.10.9 Version Table

[Table 2-9](#) provides a list of domain versions along with active current and past values for each domain version. The table lists the following properties for each domain version value:

- Anchor Version (Y/N): whether the domain version value is an anchor version or not
- Specification Version: The version of the Digital Key Specification in which that particular Domain Version value was first introduced
- Data Structure API: New data structures that were introduced in that specification version
- Notes: Informative Description of the Changes introduced in that version of the specification

Table 2-9: Active Version Table

Domain	Domain Version Value	Anchor Version? (Y/N)	Specification Version	Notes
V-OD-FW	3.0	Device : Y Vehicle : Y	TBD	- Defined MbxVer value in private mailbox to 0x01 - Set attestation package certificate version (Tag 41_h) to 0300_h (Table 11-13) - Support online BLE key retrieval from vehicle server and onto the device,

Domain	Domain Version Value	Anchor Version? (Y/N)	Specification Version	Notes
	2.0	Device : N Vehicle : N	1.2.0	- Reuse of SPAKE2+ version Tags for Owner pairing Domain Version (Table 5-4) - Defined Standard and Proprietary access profiles (Table 11-21) - Changed Certificate Version (Tag 41_h) from 01_h to 02_h (Table 11-13) - Converted Key configuration from ASN.1 to BER-TLV - Changed Tag 58_h (Entitlement Data) to Tag 78_h due to change to BER-TLV format (Table 11-13) - Removed variable updateCounter as part of BER-TLV conversion
	1.0	Device : Y Vehicle : Y	1.1.0/1.0.0	- Version was named SPAKE2+ version - Vehicle and Vehicle OEM server do not support online delivery of BLE key material. - Digital Key shared from SiaC capable device is restricted to NFC only
V-D-BT	3.0	Device: Y Vehicle: Y	TBD	- Introduced improvements to RKE related tags - Function ID 0001_h and 0002_h are deleted and replaced by 0003_h. - Introduced RKE-Hands-Free feature.
	2.0	Device : N Vehicle : N	1.2.0	- Introduced new Tags 5E_h (Table 5-4, Table 11-21) and 5F_h (Table 5-5) to track BLE/UWB versioning - Designed new SPSM Version characteristics and GATT based message exchange to communicate version information (Section 19.2.1.7 and Section 19.2.1.8) - Modified Ranging Capability RQ/RS scheme to include versioning information (Section 19.3.1.1 and Section 19.3.1.2)
	N/A	Device : Y Vehicle : Y	1.1.0/1.0.0	-
V-D-TX	1.0	Device : Y Vehicle : Y	1.2.0/1.1.0/1.0.0	Defined protocol to support existing and new Applet versions. Applet version was not incremented in version 1.2.0 of the specification. (Table 15-12 , Table 15-31)
D-VS	3.0	Device : Y Vehicle : Y	TBD	- Changed account ID hash to account Info Hash and moved to encrypted key tracking request data - Added migration metadata API
	2.0	Device : N Vehicle : N	1.2.0	- Introduced Tag 5F22_h to include Subject Key Identifier of certificate used for Signature Verification (Table 14-4) - Created new parameters in trackKey API (Section 17.7.3) and trackKey() Response (Section 17.7.7.3)

Domain	Domain Version Value	Anchor Version? (Y/N)	Specification Version	Notes
				- Introduced new API-- versionUpdate (Section 17.8.3) - Modified Unencrypted Event Notification Data (See [41]) - Added new Server Sub Status Code 50117 (See [41])
	N/A	Device : Y Vehicle : Y	1.1 1.0	-
DS-VS	3.0	Device : N Vehicle : N	1.2.0	- Negotiation of this domain version is out of scope of this specification
	N/A	Device: N Vehicle: N	1.1.0	-
OD-FD-KS	3.0	Device : Y Vehicle : Y	3.0	Sharing in a Chain
	2.0	Device : N Vehicle : N	1.2.0	- Introduced Tags 54_h (Table 11-5) and 55_h (Table 11-6) to version friend key sharing - Introduced Tags 7F2D_h and 7F11_h for Sharing Method Attestation (Table 11-10) - Modified Entitlement Data Schema to Match BER-TLV format (Table 11-12)
	N/A	Device : Y Vehicle : Y	1.1.0	-

2.10.10 Version Deprecation

Version lists in every entity should be modifiable anytime, even without a SW update. This allows to deprecate any deployed version and prevent it from being used. The mechanism to modify version lists in vehicles, servers and devices is proprietary to device and vehicle OEMs and is out of scope.

Example: Possible handling of flaw in a newer version:

- sRTR has been modified in specification between D-VS versions 2.0 and 2.2.
- sRTR processing with D-VS = 2.2 is flawed in a specific device OS version, but devices with this flaw have already been deployed. All devices also support the anchor version 2.0.
- Version lists in devices supporting this flawed version are updated to not offer version 2.2 anymore, but still support version 2.0.
- The SW update agreement process is executed (as per section 2.10.4) to inform all entities about the version update.
- Servers will now send the sRTR based on anchor version 2.0 to the device, which is able to process it correctly.
- Devices that are fixed (e.g., device OS update installed) will add back D-VS version 2.2 to their version lists so that servers can send sRTR in version 2.2 again.

- The SW update agreement process is executed one more time (as per section 2.10.4) to inform all entities about the version update.

2.10.11 Digital Key Conversion

Key conversion of existing owner and friend Digital Keys is required when vehicle and device are updated to support SiaC feature defined in this Digital Key specification. Conversion is part of full migration (see [2.10.5.3](#)). Note that on the same device there can be Digital Keys that allow sharing in a chain while others don't.

Vehicle that supports Siac feature defined in this Digital Key specification should support Digital Key conversion of Digital Keys defined in Digital Key specification 1.1.x. Successful conversion is committed to the server by sending the endpoint certificate of the converted key in the trackKey. If the key conversion is not successful on the device, then the device shall not call trackKey in this case. Vehicle server shall not send updated entitlements for new AccountRole that allow sharing in a chain in the uiBundle until the endpoint certificate of the converted key is received (see [2.10.5.3](#)).

2.10.11.1 Conversion Rules

Owner Digital Keys are converted to owner Digital Keys with sharing in a chain capability and updated configuration for shared Digital Keys, also including switching to online slot identifiers, optional immobilizer token, and online BLE Key material retrieval. Before owner device keys are converted, the owner should be notified that friend keys get extended permissions when converted.

Keys on the owner eligible devices are converted to Type 1 keys (see [Table 2-2](#)). Friend Digital Keys are converted to Type 2 or Type 3 (see [Table 2-2](#)). Friend Digital Keys shall not be converted until the owner device has been converted (see section [2.10.5.3](#)).

Successful applet update must be executed with the device SW update, before conversion is executed. The vehicle server can verify successful conversion in the new endpoint certificate that is sent in the key tracking request after conversion by checking the modified option_group_1 value ([Table 15-13](#)). The conversion occurs during full migration as illustrated in [Figure 2-13](#): and [Figure 2-14](#).

Only friend Digital Keys that have their authorized_PK list correctly populated can be converted. Therefore, Owner Devices supporting this Digital Key Specification version and legacy Digital Key Specification described in [\[41\]](#) shall provide the authorized_PK list in the key creation request to receiver devices and receiver devices shall include the authorized_PK list into their endpoints.

2.10.11.2 Applet Update

The applet supporting SiaC feature shall add support for an endpoint conversion command as described in Section [15.3.2.29](#).

2.10.11.3 *Private Mailbox Layout*

Vehicles supporting SiaC feature will require larger private mailbox sizes and a different layout relative to earlier versions of Digital Key specification to store multiple attestation packages.

Mailbox layout information is provided in owner pairing for owner Digital Keys created after vehicle and device have been SW updated to V-OD-FW agreed version 0300_h.

When an existing Digital Key is converted (see [2.10.11.1](#)), updated endpoint creation data, device configuration, and mailbox mapping are provided by the vehicle server to the device (see section [2.10.5.3](#)).

3 NFC INTERFACE [WCC1]

This section defines the NFC features that shall be implemented by the devices and how these features shall be operated for the Digital Key use case. This specification requires vehicle and device to support NFC-A technology. Support for NFC-B technology and NFC-F technology is optional on either side; corresponding requirements are only applicable if the device or vehicle is intended to support those technologies.

3.1 NFC functional requirements

This section defines the functional requirements for the NFC implementation on the vehicle and device sides.

3.1.1 *Vehicle*

The vehicle NFC readers shall be compliant with the poller requirements of the NFC Analog Technical Specification [16] for:

- Radio Frequency Power and Signal Interface
- Modulation Poller to Listener – NFC-A
- Load Modulation Listener to Poller (only for NFC-A)

If the vehicle NFC readers are intended to support NFC-B technology, they shall be compliant with the poller requirements for NFC-B as defined in NFC Analog [16]. If the vehicle NFC readers are intended to support NFC-F technology, they shall be compliant with the poller requirements for NFC-F technologies as defined in NFC Analog [16].

The vehicle NFC readers shall be compliant with the Poll Mode requirements relevant for the Type 4A Tag Platform as defined in the NFC Digital Protocol Technical Specification [17]. If the vehicle NFC readers are intended to support NFC-B technology, they shall be compliant with the Poll Mode requirements relevant for the Type 4B Tag Platform Subset as defined in NFC Digital Protocol [17]. If the vehicle NFC readers are intended to support NFC-F technology, they shall be compliant to the Poll Mode requirements relevant for the Type 3 Tag Platform Technology Subset as defined in NFC Digital Protocol [17].

The vehicle NFC readers shall be compliant with the Poll Mode requirements for the ISO-DEP protocol as defined in [17].

3.1.2 *Device*

The device NFC implementation shall be compliant with the listener requirements of the NFC Analog Technical Specification [16] for:

- The Radio Frequency Power and Signal Interface
- Modulation Poller to Listener – NFC-A
- Load Modulation Generic (only for NFC-A)
- Subcarrier Load Modulation NFC-A

The device NFC implementation may be compliant with the listener requirements for NFC-B and/or NFC-F technologies as defined in NFC Analog [16].

The device NFC implementation shall be compliant with the Listen Mode requirements relevant for the Type 4A Tag Platform defined in NFC Digital Protocol [17]. The device NFC implementation may be compliant with the Listen Mode requirements for the Type 4B Tag Platform and/or the Type 3 Tag Platform as defined in NFC Digital Protocol [17].

The device NFC readers shall be compliant with the Listen Mode requirements for the ISO-DEP protocol as defined in [17].

The device NFC implementation shall be compliant with the generic requirements for Listen Mode as defined in the NFC Activity Technical Specification [18]. The implementation also shall be compliant with the states and transitions of the Listen-Mode State Machine as defined in NFC Activity [18], which are relevant for the Type 4A Tag Platform. The device may implement additional parts of the Listen Mode State Machine, namely the states and transitions relevant for the Type 4B Tag Platform and/or Type 3 Tag Platform.

The device shall configure the Listen-Mode State Machine using the following settings:

- CON_LISTEN_T4ATP shall be enabled.

Other configuration parameter values are implementation specific. If the device is intended to listen for NFC-B technology, it shall enable CON_LISTEN_T4BTP. If the device is alternatively or additionally intended to listen for NFC-F technology, it shall enable CON_LISTEN_T3T.

The device NFC implementation shall support the following power modes:

- **Battery Operational Mode:** The battery of the device has sufficient power to support all its functions.
- **Battery Low Mode:** The battery of the device has reached a threshold at which many functions (e.g. display) are automatically disabled, but the NFC Controller function will still be powered.

The device NFC implementation shall implement means to correctly route traffic addressed to the Digital Key applet or to the Digital Key framework. Routing can be based on the application identifiers contained in the SELECT command defined in [1]. One example of such means is the routing mechanism as defined in the NFC Controller Interface Technical Specification [19], specifically the AID-based Route Selection Process.

If the device is configured to work as a Digital Key, routing to the Digital Key applet shall be enabled in Battery Operational Mode and Battery Low Mode. Additionally, routing to the Digital Key framework shall be enabled during Owner Pairing in Battery Operational Mode.

3.2 NFC Procedures

This section defines how the vehicle operates the NFC interface to detect devices and to establish and end NFC communication with a device.

3.2.1 NFC Polling and Link Setup Procedure

To turn on the RF field, the vehicle may perform RF Collision Avoidance as defined in NFC Activity [18].

The vehicle shall run a polling loop that includes the Technology Detection activity as defined in NFC Activity [18], with the following configuration settings:

- CON_POLL_A shall be enabled

Other configuration parameter values are implementation-specific. If the vehicle intends to poll for NFC-B, it shall enable CON_POLL_B. If the vehicle intends to poll for NFC-F, it shall enable CON_POLL_F.

If the Technology Detection procedure has identified one of these technologies, the vehicle NFC readers shall perform the Collision Resolution activity as defined in NFC Activity [18].

After the Collision Resolution activity, and if an NFC-A or NFC-B device indicating support for the ISO-DEP protocol as defined in NFC Digital Protocol [17], has been identified, the vehicle shall perform the Device Activation activity with the following configuration:

- INT_TECH_SEL shall be set to 000_b for NFC-A or 001_b for NFC-B
- INT_PROTOCOL shall be set to 001_b to activate the ISO-DEP protocol

Other configuration parameter values are implementation-specific. If the vehicle intends to activate a device using NFC-F technology, the vehicle NFC readers shall set INT_TECH_SEL to 010_b and INT_PROTOCOL to 100_b (Type 3 Tag Platform).

Note: The above requirements do not cover the case of multiple identified devices. If multiple devices are identified by the vehicle, the handling is implementation-specific.

3.2.2 NFC Data Transfer Procedure

The APDU exchanges defined in this specification shall be performed during the Data Exchange activity as defined in NFC Activity [18].

For devices using NFC-A or NFC-B technology, the vehicle NFC readers shall configure Data Transfer activity as follows:

- INT_PROTOCOL shall be set to 001_b to use the ISO-DEP protocol

If the vehicle intends to activate a device using NFC-F technology, the vehicle NFC readers shall set INT_PROTOCOL to 100_b (Type 3 Tag Platform).

Note: Please refer to [Appendix E](#) for details of how to operate the Digital Key contactless transactions over NFC-F.

After successful Device Activation using the ISO-DEP protocol, the vehicle shall operate the ISO-DEP protocol. The ISO-DEP protocol performs error processing in case of timeout or transmission errors. If the ISO-DEP protocol cannot re-establish communication after an error, it will raise an Unrecoverable Timeout Error or Unrecoverable Transmission Error, in which case the vehicle shall perform an NFC reset procedure.

3.2.3 NFC Link Teardown Procedure

To tear down an NFC link, the vehicle NFC readers shall perform the Device Deactivation activity as defined in NFC Activity [18].

For devices using NFC-A or NFC-B technology, the vehicle NFC readers shall configure Device Deactivation activity as follows:

- INT_PROTOCOL shall be set to 001_b to deactivate the ISO-DEP protocol

Note: Please refer to [Appendix E.5.6](#) for devices using NFC-F technology.

1 3.2.4 *NFC Reset Procedure*

2 To reset the NFC communication, the vehicle shall perform a reset of the Operating Field as
3 defined in NFC Analog [16] and shall not generate any Operating Field for a time of at least 50
4 ms.

5 Afterwards the vehicle should perform an NFC polling and link setup procedure.

6 *Note:* The NFC reset procedure uses a longer Operating Field off duration than required for other
7 NFC use cases (the minimum time for an NFC reset defined in [16] is 5.1 ms).

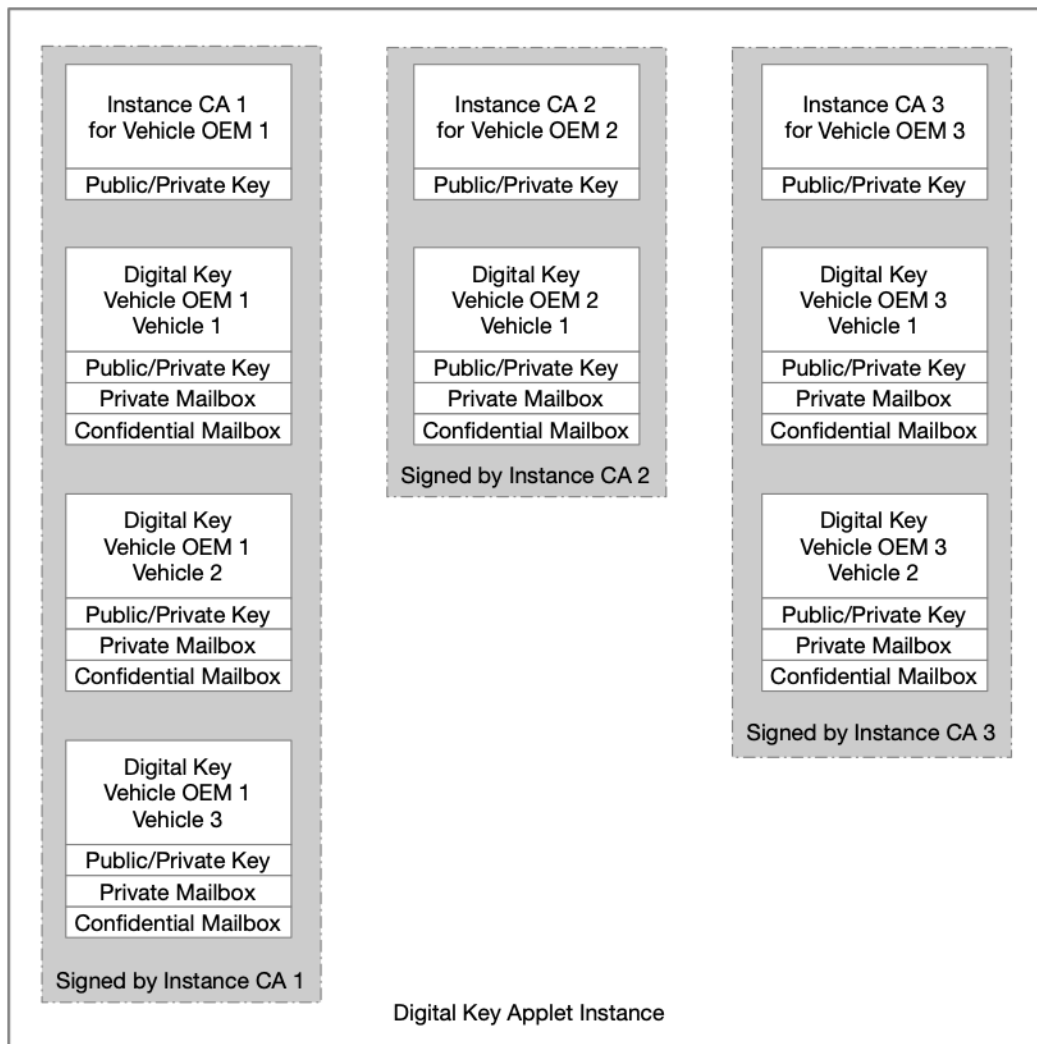
4 DATA STRUCTURE

4.1 Applet Instance Layout

One Digital Key applet instance hosts all the data necessary for a device to perform Digital Key services, as shown in [Figure 4-1](#), including all Digital Keys and one or more Instance CAs.

A Digital Key is represented by a structure and is described in [Section 4.2](#). The **Instance CAs** are intermediate CAs that represent the Device OEM CA and reside within the device. An Instance CA attests to (the public key of) a Digital Key created in the applet instance. Its signature can be verified using the Device OEM CA Certificate. A different Instance CA shall be used for each Vehicle OEM.

Figure 4-1: Applet Instance Layout



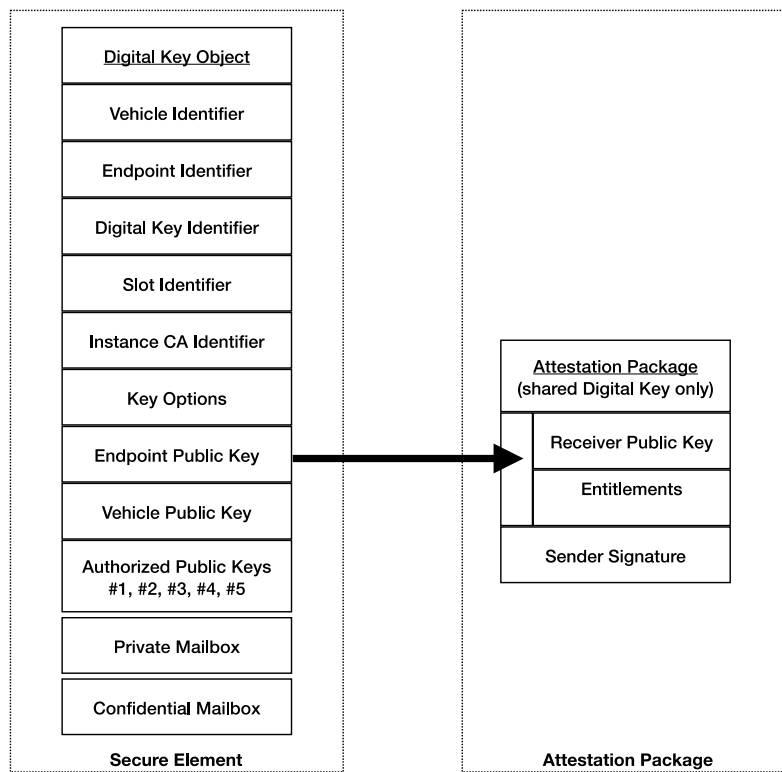
4.2 Digital Key Structure

A Digital Key object is stored in the applet instance and contains a public/private key pair, a private mailbox, a confidential mailbox, and other elements, as shown in [Figure 4-2](#).

An owner Digital Key consists of the Digital Key object only. It does not have any limitations in validity and access rights.

A shared Digital Key consists of the Digital Key object and an entitlements attestation package, which is linked to the shared Digital key by containing the same endpoint public key. The attestation package is provided and signed by the sender (device, SBOD, SBFD or KIS) and stored in the private mailbox of the receiver device with the shared Digital Key.

Figure 4-2: Digital Key Object and Entitlements Attestation Package



The elements of the Digital Key object are:

- **Vehicle Identifier (also referred to as vehicle_identifier):** Uniquely identifies the vehicle to which the Digital Key is associated. The vehicle identifier is unique per Vehicle OEM. The vehicle identifier is not identical to the VIN used in the automotive industry. It is transmitted by the vehicle during a contactless transaction.
- **Endpoint Identifier:** Used for device-internal key management. The device creates the endpoint identifier based on the rules given in [Appendix B.2](#). In [Section 15](#), the name “endpoint_identifier” is used for key creation. The identifier is reflected in the subject field of the Digital Key Certificate [H], also referred to as “Digital Key Endpoint Certificate.”

Rules associated with formatting and parsing of certificates are described in Section A.4.7 of [Appendix A](#).

- **Digital Key Identifier (also referred to as keyID):** Enables the identification of the Digital Key within the Vehicle OEM Server system. The Digital Key identifier shall be unique per Vehicle OEM. The element is named “subject key identifier” in Section [13.8](#), based on the X.509 certificate definition. It is the SHA-1 (see [\[23\]](#) and [\[24\]](#)) hash value over the device public key. See [Appendix B.2](#) for details.
- **Slot Identifier:** A value provided by the Vehicle OEM Server and used by the vehicle to identify a pre-reserved key slot for a shared Digital Key. The value is unique in the life of the vehicle and is used for replay protection and, if applicable, to identify the associated immobilizer token.
- **Instance CA Identifier:** Refers to the Instance CA that has signed the Digital Key. It is assigned by the device at Instance CA creation.
- **Key Options:** Indicates which transactions (fast, standard) the key is allowed to perform. There are several more options, such as executing a transaction over the wired interface, etc. Note that those options are not access profiles. This specification version has some modifications in the options group (see [Table 15-13](#)).
- **Endpoint Public Key:** (Also referred to as **device.PK** and **endpoint.PK** in this specification.) Used in the standard transaction. Endpoint public key shall be globally unique. It is generated at endpoint creation and is stored in the vehicle. It is represented by the “subjectPublicKey” field of the endpoint certificate in Section [15](#). Note that Device.Enc.PK (See Section [13.8](#)) is different from device public key. Similarly, device private key, device.SK, and endpoint.SK are used interchangeably in this specification.
- **Vehicle Public Key:** (Also referred to as **Vehicle.PK** and **vehicle_pk** in this specification.) Used in the standard transaction. The vehicle public key is the same for all devices associated with the same vehicle.
- **Authorized public keys:** Provided by the vehicle and are used as root in the verification chain of a receiver public key at key sharing (see Section [11](#)). In this version of the specification, the vehicle shall include only one single authorized_PK. Usage of additional authorized public keys is left for future use cases.
- **Private Mailbox:** A buffer that allows for the transfer of elements to or from the vehicle during a Digital Key Transaction (see Section [4.3.1](#) for more details).
- **Confidential Mailbox:** A buffer that allows for the transfer of elements which need confidentiality protection to or from the vehicle during a Digital Key Transaction (see Section [4.3.2](#) for more details).

All relevant data elements of the Digital Key shall be verified by the vehicle during the owner pairing procedure (see Section [6](#)) before accepting the device public key, and by the sender device during key sharing (see Section [11](#)) before signing the receiver device public key.

Before signing the receiver’s public key, the sender verifies that the receiver’s public key pair has been created on an eligible SE. The verification chain starts with an authorized public key in the sender Digital Key object, which has been provided and is trusted by the vehicle, e.g., the Vehicle OEM CA public key (see Section [16.8](#)).

The elements provided in the attestation package([Table 11-13](#)) (for shared Digital Keys only) are:

- **Receiver Public Key:** The public key created by the receiver device together with the private key. During key sharing, this public key shall be signed by the sender device together with the other fields in the attestation package.
- **Entitlements:** Below is a brief and not comprehensive list of fields present in the Entitlements data.
 - **Access Profile:** Access Profile is selected by the sender of a shared Digital Key. They need to comply with Vehicle OEM policy and are consumed/verified by the vehicle. Digital Keys that do not comply with the Vehicle OEM settings are rejected by the vehicle (i.e., their public key is not accepted). The Vehicle OEM policy is out of scope of this specification.
 - **Account Role:** Account role is selected by the sender of a shared Digital Key. Account roles govern the capabilities regarding shareability, manageability, and visibility of the shared Digital Key.
 - **Sender Key Slot Identifier:** It is the sender device Slot Identifier. The vehicle can use this to identify the sender Digital Key used for sender signature.
 - **Validity start date:** The earliest date and time the key can be used. The Digital Key may be accepted when presented to a vehicle earlier, but it shall not have effect before the date and time is reached. This field requires an internal time to be available in the vehicle. Accuracy, reliability, and security of this internal time are dependent on Vehicle OEM policy and vehicle capabilities.
 - **Validity end date:** The latest date and time the key can be used. This field requires an internal time to be available in the vehicle. Accuracy, reliability, and security of this internal time are dependent on Vehicle OEM policy and vehicle capabilities.
 - **Key Friendly Name:** A field conveying a user-friendly name for the Digital Key that should be assigned at key sharing (see [Section 11](#)). It may be the same name as in the sender's contacts when sharing a Digital Key. The sender should be allowed to edit the name for identification and personalization. For privacy reasons, friendly name should not contain private information such as full name of the friend. The same friendly name applies to keys on all devices that belong to the same device OEM account ("user"). The device OEM makes sure that the same friendly name is used. In case of conflict the first Key Friendly Name registered to the vehicle OEM server via trackKey() shall be applied for all other devices on the same account.

Digital Key elements except private and confidential mailboxes are not changeable during the lifetime of a Digital Key (exception is the conversion of an endpoint under specific conditions, see [Section 2.10.10](#)). A Digital Key can be created, terminated, and deleted. In "terminated" state, it is not usable but still able to provide the termination attestation until it is finally deleted from memory. The Digital Key states are managed internally by the applet.

4.3 Mailboxes

The mailbox mechanism allows the SE to store small data buffers accessible by the Digital Key framework and by the vehicle. These data buffers are stored in the SE's non-volatile memory and

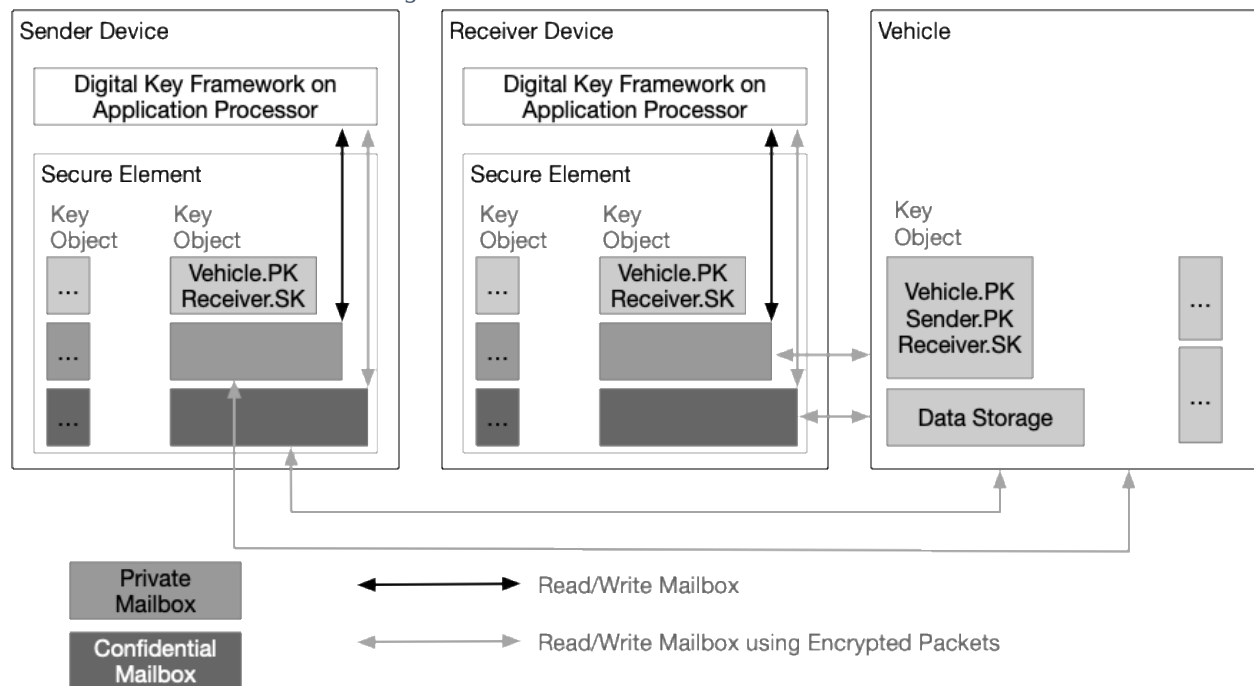
are read from/written to during transaction, key sharing, and pairing. Each Digital Key (referred to as an Endpoint) created in the applet instance of a device can have two mailbox types (private and confidential) with different security properties and targeted at different usages. The mailboxes support random access, whereby content can be read from/written to using offset/length parameters to access them.

Confidential mailboxes are not shared between Endpoints, and consequently only a vehicle paired to an Endpoint can access the confidential mailbox of such Endpoint. Private mailboxes are not shared among Endpoints, either; however, the Digital Key framework and the vehicle can freely read from and write to their contents.

The confidential mailbox is meant for storage of secret data that is never available in plaintext to the Digital Key framework. In addition, only the associated vehicle has read/write access privileges to the contents of this mailbox. The Digital Key framework can only extract or push portions of the confidential mailbox in an encrypted format. The confidential mailbox is used for storage of immobilizer tokens.

The private mailbox is meant for storage of data accessible in plaintext to the Digital Key framework, the vehicle, and the SE. The use of the private mailbox guarantees that this data is never transferred in plaintext -, and the mailbox access is only possible using the EXCHANGE command. The private mailbox is used for transfer of the attestation package and to indicate which mailbox fields are present/absent (signaling) as described in the following sections. The [Figure 4-3](#) illustrates the mailbox read/write permissions at the sender and receiver device.

Figure 4-3: Mailboxes Read/Write Permissions



4.3.1 Private Mailbox

The private mailbox can be read from and written to by the Digital Key framework using the device internal wired link as described in Section 15. It can also be read from and written to by the vehicle once the secure channel described in Section 15 is established between a vehicle and

device using the Digital Key. Data exchanged with vehicle is always protected by the established secure channel.

The mapping of the private mailbox defines data structures required by the Digital Key framework and provides a bi-directional signaling mechanism to indicate to the receiver that new data is available as described in [Figure 4-4](#).

The private mailbox enables the Vehicle OEM to define its data structures by the signaling mechanism. These data structures are opaque to the Digital Key framework and are only known to the vehicle and a Vehicle OEM app or the Vehicle OEM Server. Arbitrary data that can be stored in and retrieved from the private mailbox associated with each Digital Key includes the signaling bitmap, the slot identifier bitmap, the slot identifiers, the Vehicle OEM proprietary data, and the attestation package, in case a key is presented to a vehicle. The private mailbox format is determined at owner pairing. It shall be the same for owner and shared Digital Key mailboxes, though certain fields may not be used in shared Digital Key mailboxes.

The vehicle should configure the standard transaction to retrieve the Mailbox Version and the Signaling Bitmap in the AUTH1 command, so that the subsequent read/write commands can target the correct mailbox layout.

The Key Attestation section is used with the owner and shared Digital Key mailboxes. Note that these mailboxes have different purposes and structures. Their use for owner key is described in section 6.3.4.1.

4.3.1.1 Private Mailbox Fields

[Figure 4-4](#) shows the private mailbox sections. The Mailbox Version, MbxVer is introduced in this specification to enable the vehicle to detect which mailbox layout is present in the endpoint. Its value MBX_VERSION is defined with dependency to the agreed V-OD-FW version. This allows association of a new mailbox layout with each new V-OD-FW version or keep it independent from the V-OD-FW domain version.

The Mailbox Version is defined as the first byte of the private mailbox buffer. The MSB is always set to “1” to differentiate this layout from the one starting with the signaling bitmap in earlier Digital Key specification versions, wherein the MSB is always set to “0”. The Mailbox Version MBX_VERSION defining the layout described in this specification has a value of 80_h. The vehicle selects the mailbox layout (one or more can be allowed in future to work with the agreed V-OD-FW version) by providing the requested Mailbox Version MBX_VERSION in the mailbox mapping (see [Table 5-13](#)).

Figure 4-4: Device Private Mailbox Structure

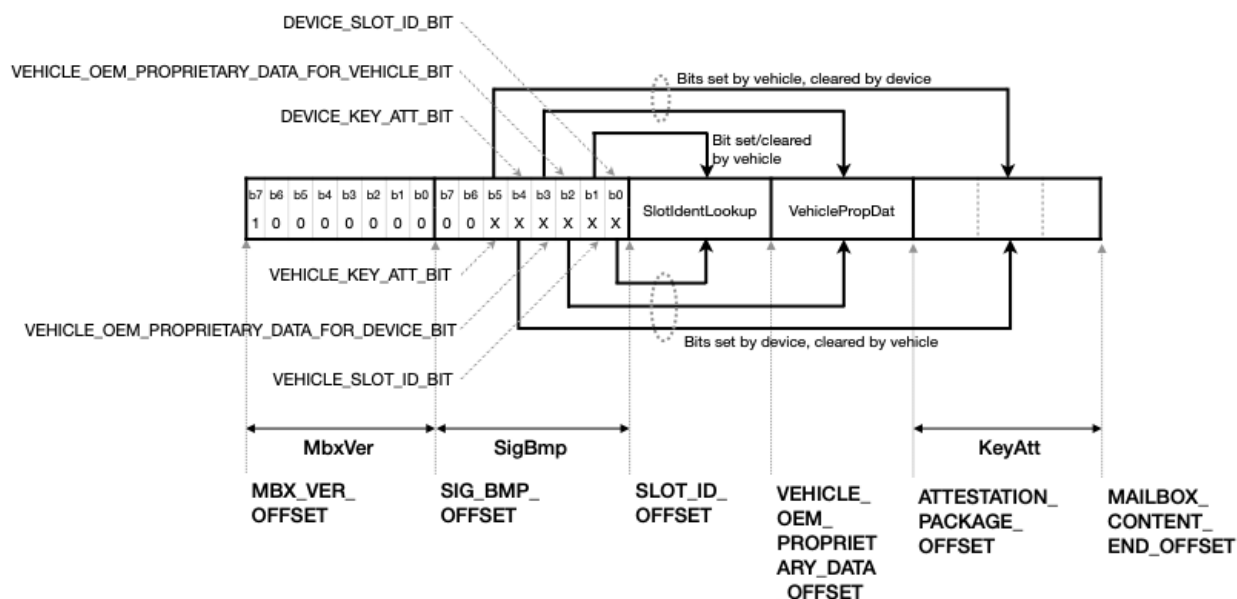


Table 4-1: Constant values for Private Mailbox

Constants	Values
DEVICE_SLOT_ID_BIT	00 _h
VEHICLE_SLOT_ID_BIT	01 _h
VEHICLE_OEM_PROPRIETARY_DATA_FOR_DEVICE_BIT	02 _h
VEHICLE_OEM_PROPRIETARY_DATA_FOR_DEVICE_BIT	03 _h
DEVICE_KEY_ATT_BIT	04 _h
VEHICLE_KEY_ATT_BIT	05 _h

The Signaling Bitmap as contained in the SigBmp field allows the vehicle and the device to inform the other party of changes occurring or actions to be taken on some mailbox field by the other party. It is the responsibility of the vehicle or the device that consumes the bit to reset the bit and delete the data if needed. The signaling relationship and the offsets of the data elements are configured by the vehicle at owner pairing (see [Table 5-13](#)).

[Table 4-2](#) shows the purpose for each bit in the SigBmp field. The signaling bits may be read and/or written by the vehicle and/or the device as defined in [Table 4-2](#). The occurrence of DEVICE_SLOT_ID_BIT and VEHICLE_SLOT_ID_BIT both set to 1 is an anomaly that occurs if the vehicle server overwrites the vehicle-provided slot identifier. This is not a defined flow but may be used to fix issues in the field. The resolution of this anomaly shall be performed by the vehicle OEM.

Devices may support this behavior or strictly enforce the configuration of tag DA_h in the sharing configuration (7F60_h). See [Table 5-14](#) for owner pairing and [Table 11-5](#) for key sharing use cases. The resolution for this anomaly is that vehicle clears (set to 0) the VEHICLE_SLOT_ID_BIT (exceptional behavior) and clears (set to 0) DEVICE_SLOT_ID_BIT.

1

Table 4-2: Signaling Bitmap Decoding

Bit	Bit Name	Description
b0	DEVICE_SLOT_ID_BIT	Set to 1 by the device to indicate to the vehicle that the device has written the Slot Identifier List in the SlotIdentLookup structure
		Cleared (set to 0) by the vehicle to indicate to the device that the vehicle has read the Slot Identifier List present in the SlotIdentLookup structure.
b1	VEHICLE_SLOT_ID_BIT	Set to 1 by the vehicle to indicate to the device that vehicle has written the Slot Identifier List in the SlotIdentLookup structure
		Cleared (set to 0) by the vehicle to indicate to the device that the vehicle has read the Slot Identifier List present in the SlotIdentLookup structure.
b2	VEHICLE_OEM_PROPRIETARY_DATA_FOR_VEHICLE_BIT	Set to 1 by the device to indicate to the vehicle that device has written vehicle proprietary data to VehiclePropDat structure
		Cleared (set to 0) by the vehicle to indicate to the device that vehicle proprietary data present in VehiclePropDat structure has been read
b3	VEHICLE_OEM_PROPRIETARY_DATA_FOR_DEVICE_BIT	Set to 1 by the vehicle to indicate to the device that vehicle proprietary data is present in VehiclePropDat structure
		Cleared (set to 0) by the device to indicate to the vehicle that vehicle proprietary data present in VehiclePropDat structure has been read
b4	DEVICE_KEY_ATT_BIT	Set to 1 by the device to indicate to the vehicle that Digital Key Attestation Package(s) is present in KeyAtt structure
		Cleared (set to 0) by the vehicle to indicate to the device that Digital Key Attestation Package(s) present in KeyAtt structure has been read
b5	VEHICLE_KEY_ATT_BIT	Set to 1 by the vehicle to indicate to the device that Digital Key Attestation Package(s) is present in KeyAtt structure
		Cleared (set to 0) by the device to indicate to the vehicle that Digital Key Attestation Package(s) present in KeyAtt structure has been read
b6-b7	RFU	Shall be set to 0 in this Digital Key specification version. RFU shall be ignored on reception by the device and vehicle, in this Digital Key specification version.

2 The SlotIdentLookup structure holds the Key_Slot_Identifier (tag 4E_h) included in each
3 Attestation Package that is present in the KeyAtt field. This allows the vehicle to check whether
4 the public key in the Attestation Package is already known or not. If the public key is already
5 known, then the vehicle does not need to read the Attestation Package and can proceed with the
6 next one and thus save time.

7 The SlotIdentLookup structure contains list of Key_Slot_Identifiers with
8 MAX_KEY_ATT_COUNT elements. Unused Key_Slot_Identifiers shall fill with FF_h. The
9 Key_Slot_Identifier order shall match the Attestation Package order in KeyAtt structure. The
10 used Key_Slot_Identifier and Attestation Package shall be packed to the start of their respective
11 fields. The size of each Key_Slot_Identifier is determined by the mailbox mapping (see [Table](#)
12 [5-13](#)).

The **Vehicle OEM proprietary data** as contained in the VehiclePropDat structure is arbitrary data specific to each Vehicle OEM. The Vehicle OEM proprietary data is present in the private mailbox until consumed and cleared by the device or by the vehicle.

If the length of this data structure is greater than 0, the Vehicle OEM proprietary data should be read by the Digital Key framework within the Digital Key termination process and sent along with the termination attestation to the KTS. The Vehicle OEM proprietary data may not be sent in all cases due to technical constraints.

The KeyAtt structure can be used for

- i) providing the Opaque Attestation (see Section 6.3.4.1) to the device (signaled by setting VEHICLE_KEY_ATT_BIT to 1),
- ii) providing the ktsSignature, indicating that the owner key was successfully registered at the KTS, to the vehicle (see Section 6.3.4.3) (signaled by setting DEVICE_KEY_ATT_BIT to 1),
- iii) for supporting Digital key sharing (providing the Attestation Package to the vehicle; see Section 11.4.7) (signaled by setting DEVICE_KEY_ATT_BIT to 1), and
- iv) resuming a suspended key using a Resume Attestation provided by the device (see Section 13.4.3) (signaled by setting DEVICE_KEY_ATT_BIT to 1).

The opaque attestation is the proof from the vehicle that the device is in possession of the private key that matches the endpoint public key in the endpoint certificate.

For shared keys, one or more Attestation Packages can be present in the KeyAtt structure. They shall be stored at offsets determined by the MAX_KEY_ATT_LENGTH (see Table 5-13). The first Attestation Package shall be stored at offset = 0, the second Attestation Package shall be stored at offset = MAX_KEY_ATT_LENGTH, the third Attestation Package shall be stored at offset = 2 x MAX_KEY_ATT_LENGTH, and so forth. For Owner keys and Shared keys the size of each attestation package (MAX_KEY_ATT_LENGTH) shall not exceed 609 bytes.

Once the key is accepted by the vehicle, the KeyAtt structure will only be used up to MAX_KEY_ATT_LENGTH. The buffer after this offset can be freed to save memory in the Secure Element, if required by the device OEM. In this case, the corresponding areas in the SlotIdentLookup structure shall be set to FE_h by the device to signal that the KeyAtt structure memory of this areas is not allocated any more. If the memory is required for future use cases, memory should be re-allocated, and the memory shall be set to 00_h. The re-allocation process is not defined in this Digital Key specification.

4.3.1.2 Private Mailbox Configuration

The sizes of the data structures in the private mailbox are defined by offsets, which are provided by the vehicle during owner pairing (see Table 5-13). Data structure sizes are calculated by subtracting the offset values. Table 4-3 describes the relationship among data elements, memory offsets, and Signaling Bitmap.

Table 4-3: Private Mailbox Content

Field	Memory Offset	Description	Sig-Bitt
MbxVer	MBX_VER_OFFSET	Mailbox version	n/a

Field	Memory Offset	Description	Sig-Bitt
SigBmp	SIG_BMP_OFFSET	Signaling Bitmap	n/a
SlotIdentLookup	SLOT_ID_OFFSET	Key_Slot_Identifier for each Attestation Package	Bits 0 and 1
VehiclePropDat	VEHICLE_OEM_PROPRIETARY_DATA_OFFSET	Vehicle OEM proprietary data structure	Bits 2 and 3
KeyAtt	KEY_ATT_OFFSET	Cryptographic key attestation package	Bits 4 and 5
MbxEnd	MAILBOX_CONTENT_END_OFFSET	Offset of first byte of free memory after last data structure	n/a

The Sig-Bit column in the above table identifies the bit of the Signaling Bitmap responsible to signal a value change in the associated data structure. The association is configured by the vehicle (see [Table 5-13](#)).

MBX_VER_OFFSET shall be set to 00_h.

SIG_BMP_OFFSET shall be set to 01_h.

SLOT_ID_OFFSET shall be set to 02_h.

The size of SlotIdentLookup data block shall be MAX_KEY_ATT_COUNT x SLOT_IDENTIFIER_LENGTH (see [Table 5-13](#)).

VEHICLE_OEM_PROPRIETARY_DATA_OFFSET defines the start position of the structure VehiclePropDat, defined by the Vehicle OEM. This structure is opaque to the Digital Key framework, and part of the structure may be disclosed to the Digital Key framework.

VEHICLE_OEM_PROPRIETARY_DATA_OFFSET is defined by the offset value in [Table 5-13](#) matching the corresponding Sig-Bit as defined in [Table 4-3](#). The VehiclePropDat structure is defined by the needs of the vehicle OEM and shall not be larger than 256 bytes.

ATTESTATION_PACKAGE_OFFSET defines the start position of the structure KeyAtt used for key sharing. KeyAtt structure size shall be MAX_KEY_ATT_COUNT x MAX_KEY_ATT_LENGTH. The buffer after this offset can be freed by Secure Element to save memory (see section [4.3.1.1](#)).

MAILBOX_CONTENT_END_OFFSET indicates the offset of first byte of the free memory after the last data structure of the private mailbox.

The order of the data structures in the mailbox buffer shall be in the order of the associated signaling bits (Sig-Bit), starting with Bit 0.

The offset values are defined together by both the Vehicle OEM and the Device OEM and are out of scope of this specification.

Note that Attestation Packages can become bigger in size in this Digital Key specification. Section [2.10.11](#) (conversion) describes how owner devices are informed about the new size requirements after a vehicle SW update.

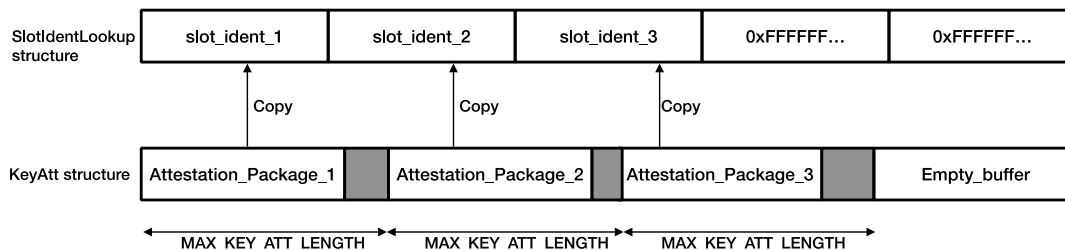
4.3.1.3 Private Mailbox Attestation Package Management

The KeyAtt structure contains zero or more Attestation Packages. The Signaling Bits for SlotIdentLookup and KeyAtt structures shall be set and cleared together in a one operation. If at least one Attestation Package is contained in the KeyAtt structure, the corresponding Key_Slot_Identifier(s) shall be present in SlotIdentLookup structure and both signaling bits shall be set to 1 in SigBmp field.

If more than one Attestation Packages are present in KeyAtt structure, then they shall be stored in the order of sharing (with the first attestation package signed by a Digital Key that is already known to the vehicle), so that the Attestation Packages are in the same position (index) as the matching Key_Slot_Identifier in the SlotIdentLookup structure. [Figure 4-5](#) is an illustration of this concept.

Unused space at the end of each Attestation Package until MAX_KEY_ATT_LENGTH shall be filled with 00_h by the device. The SlotIdentLookup structure shall be populated by the device.

Figure 4-5: Chained Attestation Package Management



Each Attestation Package in KeyAtt structure and the corresponding Key_Slot_Identifier in SlotIdentLookup structure shall be cleared (i.e., set to 00_h) by the vehicle as soon as the corresponding key is accepted by the vehicle. The signaling bits for KeyAtt and SlotIdentLookup structures shall be cleared by the vehicle if there are no more data to be read in both structures. The signaling bits for SlotIdentLookup structure and KeyAtt structure shall be cleared together in one operation. If some data could not be cleared after use, the vehicle shall check and clear up on the next usage(s) until the intended mailbox state is reached.

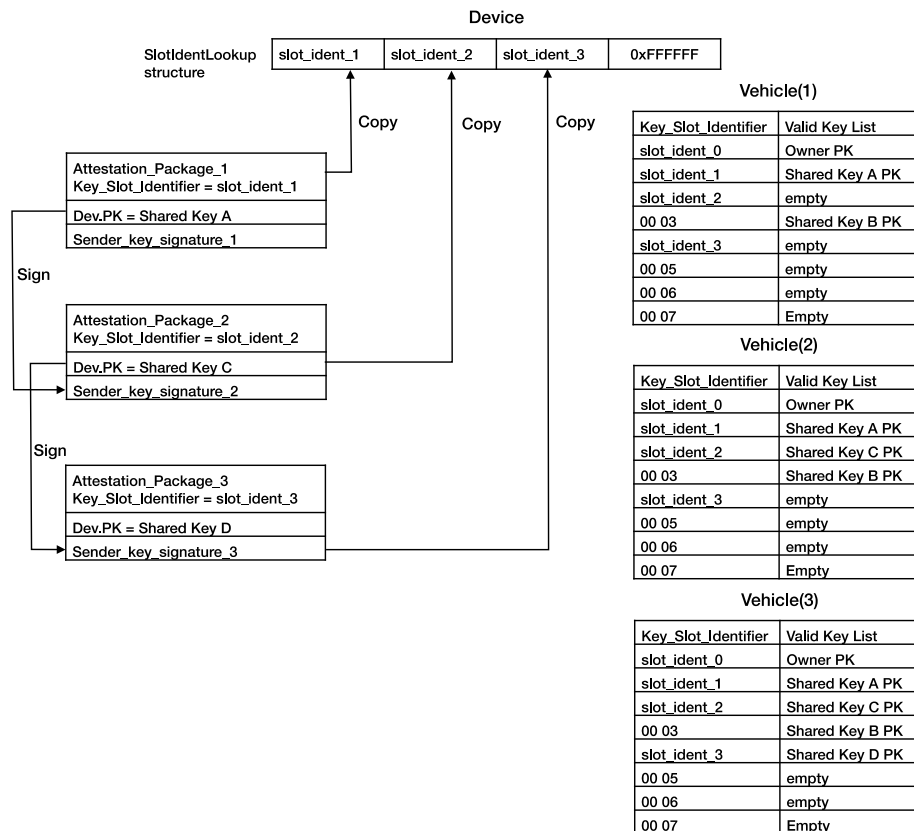
Note: For access only keys that are used via BLE/UWB there is no unlock transaction that can be used to clear the private mailbox content. In this situation, the vehicle can send the EXCHANGE command (Section [15.3.2.15](#)) to clear the KeyAtt and SlotIdentLookup data similar to the flow described in [Figure 19-21](#).

[Figure 4-6](#) illustrates an example of how Attestation Packages are validated by the vehicle.

- Vehicle(1): The vehicle has applied the algorithm described in [Listing 11-6](#) (First New Key Transaction) and found that slot_ident_2 is the first Key_Slot_Identifier that does not have an associated public key in the vehicle memory. This means that the Attestation Package with Key_Slot_Identifier = slot_ident_2 must be verified in order to store its public key in the corresponding key slot in the vehicle. The vehicle reads Attestation Package_2 and verifies its signature using the public key identified by the Sender Key_Slot_Identifier (tag 82_h) contained in the Attestation Package (see [Table 11-12](#)).

- Note that Attestation_Package_1 in the example is already known to the vehicle as it has been signed with a known public key. The vehicle does not need to read it.
- Normally the device should not provide Attestation_Package_1 in the private mailbox, but it can be delivered to the vehicle online without the device having received feedback.
- Vehicle(2): The public key of the verified Attestation_Package_2 (which is Shared Key C PK) is stored with slot_ident_2 in the vehicle.
- The vehicle looks for the next Key_Slot_Identifier without a public key, which is slot_ident_3. It reads and verifies the Attestation_Package_3 with Key_Slot_Identifier = slot_ident_3, using the key identified by the Sender Key_Slot_Identifier in the Attestation Package, i.e., Shared Key C PK.
- Vehicle(3): The public key of the verified Attestation_Package_3 (which is Shared Key D PK) is stored with slot_ident_3 in the vehicle. All octets in the next Key_Slot_Identifier are set to FF_h, which signals to the vehicle that there are no more Attestation Packages to verify. The last public key (Shared Key D PK) should be the one that is used for the current standard transaction.

Figure 4-6: Attestation Package Validation Example



4.3.2 Confidential Mailbox

The confidential mailbox can be read from and written to by the vehicle once the secure channel described in Section 7 or 8 is established between a vehicle and device using the Digital Key.

Data exchanged with vehicle is always protected by the established secure channel. The confidential mailbox can also be written to by the Digital Key framework via the internal wired link when data is encrypted with the Endpoint encryption public key. The content of the confidential mailbox may be exported in encrypted form with the encryption public key of another trusted SE if allowed by Endpoint configuration and with user consent.

If required by the Vehicle OEM, the immobilizer token may be stored and retrieved in the confidential mailbox and may be associated with each Digital Key.

The owner, sender, and receiver confidential mailboxes typically contains only one entry corresponding to its active immobilizer token. The active immobilizer token is a secret piece of data to be presented to the vehicle during a Digital Key transaction.

[Table 4-4](#) describes the confidential mailbox memory mapping for all keys when an immobilizer token is required.

Table 4-4: Confidential Mailbox Content

Memory Offset	Description
0	active immobilizer token, 0

The IMMOBILIZER_TOKEN_LENGTH is provided by the Digital Key framework. It is transmitted in the mailbox configuration table in the first NFC session (see [Table 5-13](#)).

4.3.3 Immobilizer Token

An optional Vehicle OEM requirement for the device is to store and release an immobilizer token, which is a confidential data element provided to the vehicle when it is requested (e.g., at engine start). The format and content of the immobilizer token are out of scope of this specification.

If implemented, either the vehicle (for owner only) or the Vehicle OEM Server provides the immobilizer tokens to the device. The immobilizer tokens are provided either by the Vehicle OEM Server during key tracking.

Immobilizer tokens shall be associated with a Key_Slot_Identifier, as shown in 6.3.5

4.3.4 Mailbox Access Rights

The following table summarizes access rights for the mailboxes from vehicle and Digital Key framework.

Table 4-5: Mailbox Access Rights

Mailbox	Digital Key Framework	Vehicle
Confidential	write encrypted, export encrypted	Read encrypted, write encrypted
Private	read, write	Read encrypted, write encrypted

5 OWNER PAIRING COMMANDS [WCC1/WCC2/WCC3]

This section defines the commands and data elements used for the owner pairing procedure. The owner device (framework) – vehicle version (V-OD-FW), which is exchanged in the SELECT (5.1.1) and SPAKE2+ REQUEST (5.1.2) commands covers all commands defined in Chapter 5.

The digital key applet protocol versions cover the applet transactions with the vehicle for unlock, engine start, etc., as described in sections 7, 8 and 10. The applet APIs used for endpoint creation, set confidential data, setup endpoint, etc. are managed based on the owner device (framework) – vehicle version.

5.1 Supported Commands for Owner Pairing

[Table 5-1](#) lists all commands defined for the owner pairing flow as described in [Section 6](#).

Table 5-1: Owner Pairing Command Set

Command	Instruction Byte (hex)	Implemented by
SELECT	A4	Digital Key Framework
SPAKE2+ REQUEST	30	Digital Key Framework
SPAKE2+ VERIFY	32	Digital Key Framework
WRITE DATA	D4	Digital Key Framework
GET DATA	CA	Digital Key Framework
GET RESPONSE	C0	Digital Key Framework
OP CONTROL FLOW	3C	Digital Key Framework
See Table 15-1	See Section 15.3.2	Digital Key Applet

[Table 5-2](#) lists the generic status words to be retrieved in case of error during basic input command checking, which apply to all owner pairing commands listed in [Table 5-1](#).

Table 5-2: Generic Status Words

Status Word	Comment
6700 _h	wrong length
6A80 _h	Incorrect parameters in command payload
6A82 _h	file not found
6B00 _h	wrong P1 or P2
6C00 _h	wrong Le
6D00 _h	wrong INS code
6E00 _h	wrong CLA code
9000 _h	command successfully executed

5.1.1 *SELECT Command*

5.1.1.1 *Purpose*

The vehicle sends the SELECT AID command to the device. The Digital Key framework AID is A0000008094343444B467631_h.

When the Digital Key framework is selected, the device shall respond with the data as described in [Table 5-3](#).

The device shall indicate its current pairing state to the vehicle. The possible states are:

- Not in pairing mode
- Pairing mode started and pairing password entered

The SELECT command used to select the Digital Key applet instance (using Instance AID) is described in Section [15.3.2.1](#)

5.1.1.2 *Definition*

command: 00 A4 04 00 Lc [Digital_Key_Framework_AID] 00

response: [[Table 5-3](#)]90 00

Table 5-3: Response to SELECT Command

Tag	Length (bytes)	Description	Field is
5A _h	2 × n	n supported vehicle – owner device framework versions (V-OD-FW-deviceList). (ver.high ver.low).	mandatory
5C _h	2 × m	m supported Digital Key applet protocol versions (ver.high ver.low)	mandatory
D4 _h	1	00 _h = not in pairing mode 02 _h = pairing mode started, and pairing password entered	mandatory

5.1.1.3 *Usage*

The device shall return all supported SPAKE2+ versions(renamed to V-OD-FW versions) and Digital Key applet protocol versions using two bytes per version in big-endian coding.

The reception of at least 16 versions shall be supported by all entities. An entity may transmit up to 16 versions.

Version h.l is represented as ver.high= ‘h’ (one byte) and ver.low= ‘l’ (one byte).

The versions shall be ordered from the highest to the lowest. At least one version shall be provided; otherwise the service is considered as not available.

Domain versions and their status as Anchor versions are listed in [Table 2-9](#). This includes V-OD-FW protocol versions (in addition to version 1.0 and Digital Key Applet protocol version 1.0 (coded 0100_h)) to be supported by devices and vehicles that are compliant with this version of the Digital Key Specification. Future versions of this specification may include additional versions may be listed in [Table 2-9](#). The vehicle shall match the versions with its supported versions as described in Section [6.3.3.7](#).

The device signals if it is in pairing mode or not. If the device is “not in pairing mode,” the vehicle should continue only if the vehicle is in pairing mode itself.

This command shall return a general status word as listed in [Table 5-2](#).

5.1.2 SPAKE2+ REQUEST Command

5.1.2.1 Purpose

In this command, the vehicle shall send the agreed vehicle – owner device framework version list with agreed vehicle – owner device framework (V-OD-FW-agreedVersion) first, all supported Digital Key applet protocol versions, and the Scrypt parameters of the SPAKE2+ protocol. The vehicle shall retrieve the curve point X of the SPAKE2+ protocol in the response. The parameters used in the SPAKE2+ REQUEST command are described in Section 18.

5.1.2.2 Definition

command: 80 30 00 00 Lc [Table 5-4] 00

response: [Table 5-5] 90 00

Table 5-4: SPAKE2+ REQUEST Command Fields

Tag	Length (bytes)	Description	Field is
5B _h	2 x n	Vehicle— owner device framework version list (V-OD-FW-vehicleList) (ver.high ver.low), agreed version first	mandatory
5C _h	2 x m	m supported Digital Key applet transaction (V-D-TX) protocol versions (ver.high ver.low), agreed version first.	mandatory
5E _h	2 x w	w supported BLE (V-D-BT) versions (ver.high ver.low)	Conditional. For WCC2/WCC3 capable vehicles. Shall be included if and only if agreed version in tag 5B _h is greater than 1.0 ³
7F50 _h	32	Scrypt configuration parameters	mandatory
C0 _h	16	Cryptographic salt, s	mandatory
C1 _h	4	Scrypt cost parameter, N _{scrypt}	mandatory
C2 _h	2	Block size parameter, r	mandatory
C3 _h	2	Parallelization parameter, p	mandatory
D6 _h	2	Vehicle Brand (see Table 2-1 in [35]), include this tag for all vehicles including vehicle that supports NFC only.	mandatory ⁴

Table 5-5: SPAKE2+ REQUEST Response Fields

Tag	Length (bytes)	Description	Field is
50 _h	65	Curve point X of the SPAKE2+ protocol, prepended with 04 _h as per Listing 18-3	mandatory

³ The introduction of versioning for Bluetooth subsystem (WCC2) is gated by V-OD-FW = 0100_h.

⁴ Vehicle models launched before 01-01-2021 may not support this. Vehicle models launched after 01-01-2021, including vehicle models only supporting NFC shall support this.

Tag	Length (bytes)	Description	Field is
5F _h	2 or 0	DK (V-D-BT) Protocol Version supported by device from Vehicle List, (as per Table 19-11). Tag with length 0 is returned if no commonly supported version is found.	Conditional. For WCC2/WCC3 capable devices. Present if Tag 5E _h was included in SPAKE2+ REQUEST message

5.1.2.2.1 Usage

Before sending the SPAKE2+ REQUEST command, the vehicle shall check the counter for SPAKE2+ pairing attempts. If the counter indicates 7 pairing attempts have been made, the vehicle shall not send the SPAKE2+ REQUEST command. Instead, the vehicle shall send an OP CONTROL FLOW to abort (error counter limit for wrong pairing password is exceeded, new pairing password needed (0D_h) or no specific reason(00_h)). When a new pairing password is generated, the counter shall be reset.

As described in section 2.10.7, the vehicle shall send the V-OD-FW version list with agreed V-OD-FW version first, to be used by the owner device. The vehicle shall also send the list of supported Digital Key applet protocol versions (V-D-TX), wherein the first listed version shall be used by the vehicle as the agreed version with the owner device. V-D-TX-vehicleList (Tag 5C_h) and V-OD-FW-vehicleList (Tag 5B_h) shall be included in the Key Creation Request ([Table 11-5](#)). This allows the determination of the best version to be used when key sharing with receiver device. Further, this allows the receiver device to determine the best transaction (fast transaction, standard transaction) version with the vehicle (V-D-TX) and determine V-OD-FW compatibility.

The WCC2/WCC3 capable vehicle shall send the version list (tag 5E_h) to the device in the SPAKE2+ Request. The device sends the selected version, to the vehicle in tag 5F_h. The set of V-D-BT version lists (tag 5E_h) obtained from the vehicle shall also be included in the Key Creation Request ([Table 11-5](#)). This is done because owner pairing may be first done over the NFC link and Bluetooth activation happens later. If a commonly supported BT protocol version is not found, then the device shall return empty Tag 5F_h with length 0. If the owner device does not support Bluetooth, but a receiver device does, the domain version V-D-BT becomes applicable. The process of selection of V-D-BT at the time of Bluetooth activation is shown in Figure 19-2. The version selection is described in detail in Section 19.2.1.7.

Reception of at least 16 versions shall be supported by all entities. Transmitters may transmit up to 16 versions.

Version h.l is represented as ver.high='h' (one byte) and ver.low='l' (one byte).

The first version in the list shall be the agreed version value, the following version values shall be ordered from the highest to the lowest. At least one version shall be provided; otherwise, the service is considered as not available. The vehicle shall select the highest version value supported by both the device and the vehicle as the agreed version.

The vehicle choice for the Digital Key applet protocol version is provided as part of the Digital Key applet transaction (see Section [15](#)).

All curve points of the SPAKE2+ protocol are defined following x9.63 standard [22] as the byte stream 0x04 || <x> || <y> where <x> and <y> are 32-byte integers in big-endian representation (see Section 18.1).

If the returned X value is at infinity or is not a valid point on the defined ECC curve, the vehicle shall abort the flow and shall send an OP CONTROL FLOW command with P2 value set to 0C_h as described in Section 5.1.7.

The number of Scrypt iterations (cost parameters) is a positive 4-byte unsigned integer value that configures the Scrypt function (see Section 18.4) to derive the verifier values in the Vehicle OEM Server and on the device. Other transmitted Scrypt parameters are the block size and the parallelization parameters (see Section 18.1.2). The value part of the Scrypt cost parameter TLV, Block size parameter TLV, and Parallelization TLV is encoded in big-endian format.

If the vehicle does not find any version supported by both sides for either SPAKE2+ or the Digital Key applet protocol or both, the vehicle shall send an OP_FLOW_CONTROL (“Owner Pairing Flow Control”) command with an appropriate reason code as defined in Table 5-24, instead of the SPAKE2+ REQUEST command.

If the SPAKE2+ REQUEST command is successfully processed, the vehicle and device shall calculate the shared secret K as per Listing 18-4 and Listing 18-5, respectively.

This SPAKE2+ REQUEST command may return either a general error condition as listed in Table 5-2 or one of the error conditions listed in Table 5-6. No response data shall be returned with an error status word.

Table 5-6: SPAKE2+ _REQUEST Response Error Status Words

Status Word	Comment
6985 _h	Command used out of sequence
6A88 _h	Received data invalid or zero
9484 _h	Device not ready for pairing

5.1.3 SPAKE2+ VERIFY Command

5.1.3.1 Purpose

This command mutually exchanges evidence to prove that the calculated shared secret is equal on both sides.

5.1.3.2 Definition

command: 80 32 00 00 Lc [Table 5-7] 00

response: [Table 5-8] 90 00

Table 5-7: SPAKE2+ VERIFY Command Fields

Tag	Length (bytes)	Description	Field is
52 _h	65	Curve point Y of the SPAKE2+ protocol, prepended with 04 _h as per Listing 18-2	mandatory

Tag	Length (bytes)	Description	Field is
57 _h	16	Vehicle evidence M[1]	mandatory

Table 5-8: SPAKE2+ VERIFY Response Fields

Tag	Length (bytes)	Description	Field is
58 _h	16	Device evidence M[2]	mandatory

5.1.3.3 Usage

Before sending the SPAKE2+ VERIFY command, the vehicle shall increase the counter for SPAKE2+ pairing attempts.

The vehicle shall calculate evidence M[1] as described in [Listing 18-7](#) and send it, together with curve point Y, to the device.

The device shall verify the following:

1. if the received curve point Y is a valid point on the defined ECC curve
2. the received M[1]

If both verifications are successful, the device shall calculate evidence M[2] as described in [Listing 18-8](#) and shall send M[2] back to the vehicle in SPAKE2+ Verify Response.

Only if the vehicle successfully verifies the received M[2], the vehicle shall continue the owner pairing flow.

If any of the above verification fails, i.e., an error occurs, the device shall not calculate M[2] and shall not return M[2] or any other response data except the status word. In this case, the vehicle shall send an OP CONTROL FLOW command to abort the owner pairing with P2 value set to 09_h or 00_h as described in Section [5.1.7](#), instead of the next regular command.

The SPAKE2+ VERIFY command may return either a general error condition as listed in [Table 5-2](#) or one of the error conditions listed in [Table 5-9](#).

Table 5-9: SPAKE2+ VERIFY Response Error Status Words

Status Word	Comment
6985 _h	Command used out of sequence
6A88 _h	Verification of evidence failed

The SPAKE2+ VERIFY step leads to secure channel keys used to establish a SCP03 channel between the Framework and the vehicle for the subsequent command exchanges. The used SCP03 channel is established following [Listing 18-10](#) and [Listing 18-11](#). The created secure channel keys shall be reset under the following conditions:

- After successful owner pairing
- When device returns a response other than 9000_h or 61XX_h (see [Table 5-15](#))
- A SPAKE2+ REQUEST is sent
- When a tearing of communication occurs
- When the maximum allowed time has expired without successfully terminating owner pairing

The vehicle needs to start with SPAKE2+ again to establish new keys. Note that the pairing password shall remain the same in those cases. The vehicle shall rotate the pairing password after seven failed attempts.

5.1.4 WRITE DATA Command

5.1.4.1 Purpose

This command sends all data needed to generate the Digital Key to the device. It is also used to provide an attestation from the vehicle of the device public key (PK) for key tracking purposes.

5.1.4.2 Definition

The following WRITE DATA command structure shall be used:

command: 84 D4 P1 00 Lc [command_data] [command_mac] 00

response: [response_mac] 90 00

Parameters P1 and P2 are always set to 00, except for the last WRITE DATA command, in which P1 shall be set to 80_h.

The command shall only be allowed to be sent in a secure channel.

The command may return either a general error condition as listed in [Table 5-2](#) or one of the error conditions listed in [Table 5-10](#).

Table 5-10: WRITE DATA Response Error Status Words

Status Word	Comment
6985 _h	Command used out of sequence
6A84 _h	Not enough memory

5.1.4.3 Usage

One or more WRITE DATA commands may be used to write the requested data objects into a buffer in the device.

Table 5-11: Objects for Digital Key Creation

Tag	Length (bytes)	Data Content	Field is
7F4A _h	variable	Endpoint creation data, elements from Table 15-13 , except the fields listed in the text below	mandatory
7F4B _h	variable	DER-encoded X.509 Vehicle Public Key Certificate [K], Listing 5-3	mandatory
7F4C _h	variable	DER-encoded X.509 Intermediate Certificate	optional
7F4D _h	variable	Mailbox Mapping for V-OD-FW = 0100 _h and V-OD-FW=0300 _h . See Table 5-13	mandatory
7F4E _h	variable	Device Configuration Table 5-14	mandatory
5F5F _h	0	Completion of Digital Key Data Sending	mandatory

All required objects of [Table 5-11](#) shall be written to the device in the order given in the table. One or several TLVs are allowed to be written per (sequence of) WRITE DATA command(s).

The TLV 5F5F_h shall be written at last to mark the end of the transmitted data sequence. The last WRITE DATA command of this sequence, which contains TLV 5F5F_h, shall be indicated by setting P1=80_h.

5F5F_h shall be the last TLV written to mark the end of the transmitted data sequence.

The maximum command data length shall be 239 bytes: $\text{len} = [\text{command_data}] + [\text{padding}] + [\text{command_mac}]$

$\text{len} = 239 + 1 + 8 \leq 255$ (ok)

$\text{len} = 240 + 16 + 8 > 255$ (not ok)

$[\text{command_data}] + [\text{padding}]$ shall be a multiple of the AES block size (16 bytes). The padding scheme is described in [9]. At least one byte padding (80_h) shall be required. The maximum response data length shall be 239 bytes.

The Tags in Table 5-11 are described below.

1. Endpoint Creation Data in 7F4A_h

The endpoint configuration data Table 5-12 shall contain all endpoint configuration data for every V-OD-FW domain version that is supported by the vehicle, so that shared devices that support different V-OD-FW versions than the owner device can create their endpoint appropriately.

Tag 60_h groups data elements in endpoint configuration data corresponding to MbxVer value 80_h. This allows the vehicle to provide endpoint configuration data corresponding to devices supporting only [41] in a backward compatible manner. All data elements under Tag 60_h in Tag 7F4A_h overwrite the data elements provided in Tag 7F4A_h (not under Tag 60_h) if the device supports V-OD-FW version 0300_h.

Device creating an endpoint with V-OD-FW agreed version to 0300_h shall always set bit 4 and bit 5 in option_group_1 to 1 in the endpoint configuration (see Table 15-13). Device creating an endpoint with V-OD-FW agreed version less than 0300_h shall set bits in option_group_1 as indicated in 46_h in 7F4A_h in the endpoint configuration (see Table 15-13).

Note: It is recommended that Future Digital Key specification version define tag 61_h corresponding to MbxVer 81_h if there is a need to carry more information in Table 5-12.

Table 5-12 lists the data elements from Table 15-13 (not including the outer tag 7F27_h), except for the following elements:

- Endpoint identifier (defined by device)
- Instance CA identifier (defined by device)
- Account info hash (defined by device)

If the vehicle_identifier is provided as part of Tag 7F4A_h, the device shall compare it to the value provided in the Vehicle Public Leaf Certificate [K] and abort the owner pairing if the check fails.

In this version of the specification, the vehicle shall include only one single authorized_PK (inner Tag 49_h). This authorized_PK shall correspond to the Vehicle intermediate CA or the Vehicle OEM CA PK (root).

1

Table 5-12: Endpoint Configuration Data

Tag			Length (Bytes)	Description	Field Is
7F4A _h			Variable	Endpoint creation data for V-OD-FW = 0100 _h and 0300 _h , unless otherwise specified	mandatory
	4D _h		8	vehicle_identifier	mandatory
	46 _h		1	option_group_1 (on bits 0 – 7)	mandatory
	47 _h		1	option_group_2 (on bits 0 – 7)	mandatory
	5C _h		2	protocol_version.	mandatory
	5B _h		65	vehicle_PK prepended by 04 _h	mandatory
	51 _h		15 or 13	not_before, DER encoded GeneralizedTime (15 bytes in length) or UTCTime (13 bytes in length) as per RFC 5280 [2]	mandatory
	52 _h		15 or 13	not_after, DER encoded GeneralizedTime (15 bytes in length) or UTCTime (13 bytes in length) as per RFC 5280 [2]	mandatory
	49 _h		variable	authorized_PK[], list up to 5 public keys prepended by 04 _h of 65 bytes	Optional
	4A _h		2	confidential_mailbox_size	Optional
	4B _h		2	private_mailbox_size	Optional
	4E _h		1-8	Key_slot	Optional
	60 _h		variable	Endpoint creation data for V-OD-FW = 0300 _h . Following data elements are present only if their values are different from values listed under 7F4A _h and outside tag 60 _h .	Optional
		D5 _h	1	MBX_VERSION = 80 _h	Optional
		4A _h	2	confidential_mailbox_size	Optional
		4B _h	2	private_mailbox_size	Optional

2

32. Vehicle Public Key in 7F4B

4The vehicle public key shall be provided as a X.509 certificate signed by the Vehicle OEM CA

5as described in Listing 5-3.

6The X.509 [3] definition of the basic constraints extension is described in Listing A-5.

7The X.509 [3] definition of the key usage extension is described in Listing A-4.

8

Listing 5-1: Vehicle Certificate Extension Schema

```
1 VehicleCertificateExtensionSchema ::= SEQUENCE
2 {
3   extension_version INTEGER (1..255)
4 }
```

9

Listing 5-2: Vehicle Certificate Extension Data

```
1 vehicle-cert-extension-data VehicleCertificateExtensionSchema ::=
2 {
3   extension_version 1 --value shall be 1
4 }
```

10The Vehicle Public Key Certificate data is described in Listing 5-3. See [3] for details on X.509

11v3 certificate format.

1

Listing 5-3: Vehicle Public Key Certificate Data

```
1 vehicle-key-cert-data Certificate ::=
2 {
3   tbsCertificate
4   {
5     version v3, --shall be v3--
6     serialNumber ..., --a random integer chosen by the certificate issuer,
7     Signature
8     {
9       algorithm {1 2 840 10045 4 3 2}--OID for ecdsaWithSHA256 (ANSI X9.62 ECDSA algorithm with SHA-256)
10    },
11    issuer rdnSequence:
12    {
13      {
14        {
15          type {2 5 4 3}, --OID for CommonName
16          value".."--shall match the subject of the issuing certificate, shall use PrintableString or UTF8String format
17        }
18      }
19    },
20    validity
21    {
22      notBefore Time:".."--shall use UTCTime or GeneralizedTime as defined in [3]
23      notAfter Time:".."--shall use UTCTime or GeneralizedTime as defined in [3]
24    },
25    subject rdnSequence:
26    {
27      {
28        {
29          {
30            type {2 5 4 3}, --OID for CommonName
31            value"Vehicle OEM Identifier"--contains the subject of the certificate, as per Appendix B.2.6
32            shall use PrintableString or UTF8String format
33          }
34        }
35      },
36      subjectPublicKeyInfo
37      {
38        algorithm
39        {
40          algorithm {1 2 840 10045 2 1}--OID for ecPublicKey (ANSI X9.62 public key type)
41          parameters {1 2 840 10045 3 1 7}--OID for prime256v1(ANSI X9.62 named elliptic curve)
42        },
43        subjectPublicKey'04..'H--the public key pre-pended with 04h to indicate uncompressed format
44      },
45      extensions
46      {
47        {
48          extnID {1.3.6.1.4.1.41577.5.1}, --OID for Vehicle Public Key Certificate (see Appendix B.2.2)
49          critical TRUE,
50          extnValue '...'H --DER encoding for VehicleCertificateExtensionSchema extension as per Listing 5-1
51        },
52        {
53          extnID {2 5 29 15}, --KeyUsage standard extension
54          critical TRUE,
55          extnValue '03020780'H --DER encoding for KeyUsage, digitalSignature only
56        },
57      }
```

```

58     extnID    {2 5 29 19}, --BasicConstraints standard extension
59     critical   TRUE,
60     extnValue '300'H -- DER encoding for CA=FALSE
61 },
62 {
63     extnID    {2 5 29 35}, --OID for AuthorityKeyIdentifier standard extension
64     critical   FALSE,
65     extnValue '..'H- DER encoding of an AuthorityKeyIdentifier sequence, containing only a KeyIdentifier element.
66 -- The KeyIdentifier is an OCTET STRING containing the 160-bit SHA-1 hash of the value of the BIT STRING
    subjectPublicKey
67         --from the issuer certificate (excluding the tag, length, and number of unused bits)
68     },
69     {
70         extnID {2 5 29 14}, -- OID for SubjectKeyIdentifier standard extension
71         critical FALSE,
72         extnValue '...'H --160-bit SHA1 hash of the value of the BIT STRING subjectPublicKey
            --(excluding the tag, length, and number of unused bits)
73     }
74 },
75 signatureAlgorithm
76 {
77     algorithm {1 2 840 10045 4 3 2}
78 },
79 signatureValue'..'H --the certificate signature computed as per [3]
80 --ECDSA signature
81 }

```

- 1 The subject field is formatted as described in Appendix [B.2.3](#)
- 2 The ROUTING_INFORMATION (as used in [11.6](#)) is defined in the common name of the
- 3 subject field of [Listing 5-3](#), i.e., the subject ID of the Vehicle Public Key Certificate [K]
- 4 (Appendix B.2.6).
- 5 3. *Intermediate Certificate in 7F4C*
- 6 The Vehicle OEM intermediate certificate (see [B.2.7](#)) may be required by the Vehicle OEM if
- 7 the Vehicle OEM CA is not able to sign the Vehicle Public Key Certificate directly.
- 8 4. *Mailbox Mapping in 7F4D_h*

Table 5-13: Mailbox Mapping

Tag		Length (bytes)	Description	Field is
7F4D _h		variable	Mailbox mapping for V-OD-FW = 0100 _h	mandatory
	D0 _h	2	SIGNALING_BITMAP_OFFSET as per Table 4-3 in [41]	mandatory
	D1 _h	2	MAILBOX_CONTENT_END_OFFSET as per Table 4-3 in [41]	mandatory
	D2 _h	1	IMMOBILIZER_TOKEN_LENGTH as per [41]	conditional
	C0 _h	2	Sig-Bit 0: corresponding data offset for SlotIdentBmp as per Table 4-3 in [41]	mandatory
	C1 _h	2	Sig-Bit 1: corresponding data offset for SlotIdentLst as per Table 4-3 in [41]	conditional

Tag			Length (bytes)	Description	Field is
	C2 _h		2	Sig-Bit 2: corresponding data offset for VehiclePropData as per Table 4-3 in [41]	conditional
	C3 _h		2	Sig-Bit 3: corresponding data offset for KeyAtt as per Table 4-3 in [41]	conditional
	60 _h		variable	Mailbox configuration for V-OD-FW = 0300 _h	Optional
		D0 _h	2	SIG_BMP_OFFSET as per Table 4-3	mandatory
		D1 _h	2	MAILBOX_CONTENT_END_OFFSET as per Table 4-3	mandatory
		D2 _h	1	IMMOBILIZER_TOKEN_LENGTH as per section 4.3.2	conditional
		D3 _h	1	SLOT_IDENTIFIER_LENGTH as per section 4.3.1	mandatory
		D4 _h	2	MAX_KEY_ATT_LENGTH as per section 4.3.1	mandatory
		D5 _h	1	MBX_VERSION = 80 _h	mandatory
		C0 _h	2	Sig-Bit 0 and 1: corresponding data offset for SlotIdentLookup as per Table 4-3	mandatory
		C1 _h	2	Sig-Bit 2 and 3: corresponding data offset for VehiclePropDat as per Table 4-3	mandatory
		C2 _h	2	Sig-Bit 4 and 5: corresponding data offset for KeyAtt as per Table 4-3	mandatory

The mailbox mapping Table 5-13 shall contain all mailbox mapping data for every V-OD-FW domain version that is supported by the vehicle, so that shared devices that support different V-OD-FW versions than the owner device can create their mailbox using the appropriate layout.

Tag 60_h groups mailbox mapping corresponding to MbxVer value 80_h (see section [4.3](#)). This allows the vehicle to provide the mailbox mapping corresponding to devices supporting only [\[41\]](#) in a backward compatible manner (no sub-tag under 7F4D_h).

Note: It is recommended that Future Digital Key specification version define tag 61_h corresponding to mailbox version 81_h, if there is a need to carry more information in [Table 5-13](#).

The entire mailbox mapping tag 7F4D_h shall be provided to the receiver device during Digital Key sharing in Import Request (see [Table 11-11](#)) because the receiver may further want to share Digital key to a device with a different V-OD-FW domain version.

Conditional fields shall be present when the data element designated by the corresponding Sig-Bit is used.

5. Device Configuration Data in 7F4E_h

[Table 5-14](#) provides all device configuration data to the device. The device configuration data [Table 5-14](#) shall contain all device configuration data for every V-OD-FW version that is supported by the vehicle, so that shared devices that support different V-OD-FW versions than the owner device can configure device data appropriately.

Tag 60_h groups data elements in device configuration data corresponding to Mb_xVer value 80_h. This allows the vehicle to provide device configuration data corresponding to devices supporting only [\[41\]](#) in a backward compatible manner.

Note: It is recommended that Future Digital Key specification version define tag 61_h corresponding to Mb_xVer 81_h, if there is a need to carry more information in [Table 5-14](#).

Table 5-14: Device Configuration Data

Tag			Length (bytes)	Description	Field is
7F4E _h			variable	Device configuration data for V-OD-FW = 0100 _h , unless otherwise specified	mandatory
	7F49 _h		variable	7F49 _h template is used to provide wireless capability (NFC/UWB) and related data; this template shall be included by vehicle. If this tag is not present, only NFC is supported. (Equivalent to only 5F50 tag present) Tags D3 _h and D4 _h shall not be included if this Tag is transmitted during Owner Pairing. See Table 19-90 for details	optional
	7F60 _h		variable	SHARING_CONFIGURATION (for key sharing, see Section 11)	mandatory
		DA _h	1	01 _h immobilizer token and slot identifiers are retrieved online 03 _h no immobilizer token, and slot identifiers are retrieved online 04 _h immobilizer token and slot identifiers for owner are retrieved from vehicle and receiver immobilizer token and slot identifiers are retrieved online. If this setting is used during owner pairing, the owner device shall use 01 _h for sharing. 05 _h no immobilizer token but with slot ID retrieved from vehicle for owner device; no immobilizer token but with slot ID retrieved online for receiver device. If this setting is used during owner pairing the owner device shall use 03 _h for sharing. 00 _h and 02 _h are deprecated and shall not be used in this Digital Key specification.	mandatory
		DB _h	0	If this tag is present, all devices have to obtain a key tracking receipt (owner during Phase 4 as per Section 6.3.5 , and receiver during Step 7 as per Section 11.4.3)	mandatory
		DC _h	0	If this tag is present, vehicle manufacturer supports online attestation package delivery feature for key sharing as described in Section 11.10 .	optional

Tag			Length (bytes)	Description	Field is
		DD _h	1	MAX_KEY_ATT_COUNT Number of attestation packages that can be verified in one First New Key Transaction, including the one for the new key itself. The minimum value is 2.	Mandatory
		DE _h	variable	If this tag is present, the vehicle supports key update attestations. Further information can be coded into the value in future Digital Key specifications. In this Digital Key specification, this tag shall be absent, but the devices shall not fail if the tag is present in future.	optional
	D7 _h		1	2 nd Factor activation required (00 _h =not required, 01 _h =required for non-approved sharing methods, other=RFU)	optional
	D8 _h		1	SHARING_PASSWORD_LENGTH (vehicle OEM specific as per appendix B.1. Present only if 2 nd factor activation is required. i.e., Tag D7 _h is present and set to 01 _h . If absent, sharing password (Section 11.2.1.4) is not required. Also signifies length of online sharing PIN.	optional
	D9 _h		variable	Supported Digital Key Access Profile list as described in Section 11	optional
	Tag 4A _h and 4B _h of Table 15-54 (SETUP ENDPOINT Command Payload)				optional
	9F20 _h		9	Content of HU_PP message (Table 19-65)	optional
	60 _h		variable	Device configuration data for V-OD-FW = 0300 _h . Following data elements are present only if their values are different from values listed under 7F4E _h and outside tag 60 _h .	Optional
		D5 _h	1	MBX_VERSION = 80 _h	Optional
		Tag 4A _h and 4B _h of Table 15-54 (SETUP ENDPOINT Command Payload)			Optional

- 1
- 2 If tag 7F60_h is absent, then owner pairing shall fail.
- 3 If tag D7_h is absent, then no second factor activation is required.
- 4 **Note:** Information in Tag D7_h could be overwritten by the presence of the activationRequired
- 5 field in the trackKey() response (17.7.3.3) and/or eventNotification (see 17.9.1).
- 6
- 7 The supported key profiles list defines the key profiles for key sharing that are supported by the
- 8 vehicle. The profiles are defined as a list of 1-byte enumeration values corresponding to Table
- 9 11-20.
- 10 If neither tag 4A_h nor 4B_h is included in the Device Configuration Data sent by vehicle to device
- 11 using WRITE DATA command (see step 3 in Figure 6-3), then device shall transmit no data in
- 12 the AUTH1 response to the vehicle (see step 6 in Figure 6-10 and Section 6.3.4.2).

If tag 4A_h or 4B_h or both are included in the Device Configuration Data sent by vehicle to device using WRITE DATA command (see step 3 in [Figure 6-3](#) then device shall transmit the corresponding data in the AUTH1 response to the vehicle (see step 6 in [Figure 6-8](#)).

5.1.5 GET DATA Command

5.1.5.1 Purpose

This command shall continue to use the established session keys to retrieve all data needed to verify the Digital Key created by the Digital Key framework in the Digital Key applet instance.

5.1.5.2 Definition

command: 84 CA 00 00 Lc [encrypted_tag] [command_mac] 00

response: [response_payload] [response_mac] 90 00 or 61XX

Only one tag shall be requested per GET DATA command. The tags may be requested in arbitrary order. Elements may be omitted if they are not required.

X.509 certificates shall not be wrapped into additional TLV structures, but tags shall be used to refer to the certificates in the command tag list, as indicated below.

Le shall be set to 00_h as the exact length of data depends on the variable certificate field sizes. GET RESPONSE shall be used if the response status word indicates that more data is to be retrieved.

This command may return either a general error condition as listed in [Table 5-2](#) or one of the error conditions listed in [Table 5-15](#).

Table 5-15: GET DATA Response Error Status Words

Status Word	Comment
61XX _h	XX indicates the number of response bytes still available (use GET RESPONSE)
6985 _h	Command used out of sequence
6A88 _h	Reference data not found
6400 _h	Digital Key could not be created
6402 _h	Requested elements not available

5.1.5.3 Usage

The command requests the transfer of the required data element by providing the corresponding tag. The response data length shall not exceed 239 bytes, as the maximum length of an APDU response shall not exceed 258 bytes (including status word):

len = [response_data] + [padding] + [response_mac] + [status_word]

len = 239 + 1 + 8 + 2 ≤ 258 (ok)

len = 240 + 16 + 8 + 2 > 258 (not ok)

[response_data] + [padding] equals a multiple of the AES block size (16 bytes). At least one byte padding (80_h) is required. The maximum response data length is then 239 bytes.

See Section 18.4.12 for calculation of [command_mac] and [response_mac].

Each element shall be requested in a separate GET DATA command. GET RESPONSE shall be used if the returned data exceeds the GET DATA response data maximum length.

When the response is a TLV structure (e.g., certificate) then the response shall not be wrapped in the requested tag. Otherwise (e.g., string), the response shall be wrapped in the requested tag.

If all the remaining bytes cannot be sent in the current response, the status word 61XX_h shall be used instead of 9000_h.

If XX is between 01_h and EF_h, XX number of bytes (1..239) are still available and shall be sent in the next response (which shall be the last one). Note that XX does not include padding length.

If XX equals 00_h , not all remaining bytes can be sent in the next response.

Values between F0_h and FF_h are not allowed for XX.

Table 5-16: GET DATA Response Decrypted Payload for tag 7F20_h

Description
Vehicle OEM signed Device OEM CA Certificate (DeviceOEM.PK.VehicleOEMCert), [F] as per Listing 15-12

Table 5-17: GET DATA Response Decrypted Payload for tag 7F22_h

Description
Device OEM signed Instance CA Certificate, [E] as per Listing 15-15

Table 5-18: GET DATA Response Decrypted Payload for tag 7F24_h

Description
Digital Key Certificate signed by Instance CA (DigitalKey.DKCert) as per Listing 15-5

Table 5-19: GET DATA Response Decrypted Payload for Tag D3_h

Tag	Length (bytes)	Description
D3 _h	1–30	Friendly name of the owner key (UTF-8) with length of 1 to 30 bytes

The friendly name of the owner Digital Key should be assigned during owner pairing. The friendly name shall be a UTF-8 encoded string. The owner should be allowed to edit the name for identification and personalization. For privacy reasons, the friendly name should not contain private information, such as the full name of the owner.

5.1.6 GET RESPONSE Command

5.1.6.1 Purpose

This command shall only be used to retrieve remaining data that could not be fully transmitted in the GET DATA response. This command shall only be accepted if the GET DATA response received previously was with a status word 61XX_h

5.1.6.2 Definition

command: 84 C0 00 00 Lc [command_mac] 00
response: [response_payload] [response_mac] 90 00 or 61XX

When the response data is not fully transmitted in the response payload of GET RESPONSE, a status word 61XX_h shall be returned. XX_h indicates the number of response bytes still available. When a response data is completely transmitted in the response payload of GET RESPONSE and no more remaining data is to be returned, a status word 9000_h is returned. This command may return either a general error condition as listed in [Table 5-2](#) or one of the error conditions listed in [Table 5-20](#).

Table 5-20: GET RESPONSE Response Error Status Words

Status Word	Comment
6985 _h	Command used out of sequence

5.1.6.3 Usage

One or more GET RESPONSE commands shall only be used to retrieve all response data that could not be fully transmitted in the GET DATA response. The command shall only be accepted when sent immediately after a GET DATA or GET RESPONSE response with the status word 61XX_h. The vehicle shall send the last GET RESPONSE command when a GET RESPONSE response with XX of 1..239 is received. Values between F0_h and FF_h are not allowed for XX. When XX is 00_h, at least one more GET RESPONSE command shall follow.

5.1.7 OP CONTROL FLOW Command

5.1.7.1 Purpose

With this command, the vehicle may control the flow at predefined points and abort the flow at any time while giving a status to the device. “Continue” and “End” shall be used where specified in the flow diagrams. Status information about the state, successful or unsuccessful end, etc. shall be provided.

5.1.7.2 Definition

This command is always sent outside of the secure channel.

command: 80 3C [[Table 5-21](#)][[Table 5-22](#) or [Table 5-23](#) or [Table 5-24](#)]

response: 90 00

This command may return a general error condition, as listed in Table 5-2.

5.1.7.3 Usage

The vehicle shall not consider the session as terminated before it has received the response to the OP CONTROL FLOW command (applicable for P1=11_h or P1=12_h).

The OP CONTROL FLOW response should be kept as short as possible to minimize the risk of false positives when a tearing happens on the response.

When aborted, the vehicle shall perform the NFC link teardown procedure, followed by the NFC reset procedure, before it restarts owner pairing or any other transaction.

P1 indicates the main action triggered by the command; P2 provides the reason code.

- 1 The Digital Key Framework shall ignore unknown P1/P2 values, if a P1 value is not known it
2 shall be treated as P1=10_h (continue flow) and ignored.

3 *Table 5-21: OP CONTROL FLOW P1 Parameters*

P1 Value	Description
10 _h	continue flow, see reason code for details
11 _h	end flow, see reason code for details
12 _h	abort flow, see reason code for details

4 *Table 5-22: OP CONTROL FLOW P2 Parameters for P1=10_h (continue)*

P2 Value	Description
01 _h	key creation data transmitted to device
02 _h	key certificate chain received by vehicle
03 _h	Waiting for user confirmation on vehicle UI
0F _h	Vehicle waiting time extension, keep busy, no action on device side

5 *Table 5-23: OP CONTROL FLOW P2 Parameters for P1=11 (end with success)*

P2 Value	Description
11 _h	successful end of key creation and verification, key not tracked by vehicle

6 *Table 5-24: OP CONTROL FLOW P2 Parameters for P1=12 (end with failure)*

P2 Value	Description
00 _h	no specific reason
01 _h	no matching SPAKE2+ version found between vehicle and device
02 _h	no matching Digital Key applet protocol version found between vehicle and device
03 _h	pairing failed due to timeout of timer 1
04 _h	pairing failed due to timeout of timer 2
05 _h	pairing failed due to timeout of timer 3
06 _h	pairing failed due to timeout of timer 4
07 _h	pairing failed due to timeout of timer x (spare, not used)
08 _h	preconditions for owner pairing not fulfilled
09 _h	evidence verification on vehicle side failed
0A _h	wrong Digital Key configuration
0B _h	certificate verification failed
0C _h	curve point X zero or invalid
0D _h	error counter limit for wrong pairing password is exceeded, new pairing password needed
0E _h	No matching SPAKE2+ version found between vehicle and device because at least one key with higher SPAKE2+ version exists.
7F _h	pairing failed due to response data or format error of previous command

P2 Value	Description
B8 _h	Cannot perform owner pairing because the vehicle is in drive ready state.

The option to end the flow refers to the regular end of the transaction and the possible outcomes (successful, not successful) and the UI actions.

The option to abort the flow refers to the vehicle aborting the flow before its regular end due to unexpected or erroneous behavior.

5.2 Data Elements

5.2.1 Certificates

Table 5-25: Certificates

Tag	Certificate	Reference
7F20 _h	Device OEM or SBxD/KIS Certificate signed by Vehicle OEM CA	Listing 15-12
7F22 _h	Instance CA Certificate signed by Device OEM CA	Listing 15-15
7F24 _h	Digital Key Certificate signed by Instance CA	Listing 15-5
7F4B _h	Vehicle Public Key Certificate	Listing 5-3
7F4C _h	Vehicle OEM Intermediate Certificate	optional, (Vehicle OEM specific)
7F42 _h	SBxD/KIS Intermediate Certificate	Listing 11-13
7F44 _h	SBxD/KIS Endpoint Certificate	Listing 11-10

The Vehicle OEM intermediate certificate may be required by the Vehicle OEM if the Vehicle OEM CA is not able to sign the Vehicle Public Key Certificate directly.

The external CA certificate (7F20_h) follows the same rules for device OEMs and SBxD/KIS.

The SBxD/KIS intermediate certificate (7F42_h) is based on the instance CA certificate structure but is adapted to its purpose of being an online CA.

The SBxD/KIS endpoint certificate (7F44_h) is based on the endpoint certificate structure and adds an additional extension for server-specific use cases. It shall contain a revocation check method (CRL [\[3\]](#), OCSP [\[44\]](#), etc.)

Refer to section [16.2.14](#) for details about the certificate structures.

6 OWNER PAIRING

6.1 Overview

The owner pairing flow is operated by the Digital Key framework running on the device. As shown in [Figure 2-5](#), the Digital Key framework uses the APDU commands described in Section 5 to manage the configuration of the Digital Key, protected by the SE.

The SE operating system provides the root of trust, which is the starting point in the trust chain as defined in Section 16.1.

The vehicle is able to select the Digital Key applet over NFC using its AID and to select the Digital Key framework using the corresponding AID. The NFC controller may be reconfigured for changing the routing of the communication from the SE to the Digital Key framework and vice versa, based on the selected AID.

A new owner device pairing flow or owner device change does not imply an implicit unpairing, i.e., a new device owner pairing flow only changes the owner's key. Existing shared keys that are already paired and vehicle public keys are not necessarily impacted.

It shall not be possible to change the owner device to another device that has a lower V-OD-FW version than the preceding owner device as long as digital keys for the vehicle still exist. To achieve this, the vehicle shall not indicate support of V-OD-FW=0100_h during owner pairing if there exist Digital Keys for the vehicle that are based on V-OD-FW=0300_h. If a device that supports only V-OD-FW=0100_h is attempting owner pairing and the preceding owner device with V-OD-FW=0300_h is not unpaired first, then the vehicle shall send a OP CONTROL FLOW P1 = 12_h and P2 = 0E_h (no matching SPAKE2+ version found between vehicle and device because at least one key with a higher SPAKE2+ version exists) instead of a SPAKE2+ Request command.

Once the vehicle is paired using version 0300_h, the vehicle shall support V-OD-FW=0100_h only after deletion of all existing keys (via unbinding, unpairing, "reset" function in the vehicle, remote deletion, local deletion, etc.)

The Digital Key applet Instance shall be available on the SE before the time of owner pairing execution.

6.2 Keys and Data

This section defines the key and data elements used in owner pairing:

- **device.PK/device.SK**: Long term key pair generated by device during Digital Key creation. One per Digital Key.
- **vehicle.PK/vehicle.SK**: Key pair generated by the vehicle. Same for all Digital Keys of the vehicle. The lifecycle of this key pair is defined and managed by the Vehicle OEM and is out of scope of this specification.
- **Pairing password**: SPAKE2+ pairing password entered into the device; eight numeric digits (0–9) in UTF-8 format provided by the Vehicle OEM account to authenticate the owner.
- **Kenc**: Derived symmetric key (from SPAKE2+ shared secret), used to encrypt confidential command and response payloads.

- **Kmac**: Derived symmetric key (from SPAKE2+ shared secret), used to calculate command MACs.
- **Krmac**: Derived symmetric key (from SPAKE2+ shared secret), used to calculate response MACs.

6.3 Owner Pairing Implementation

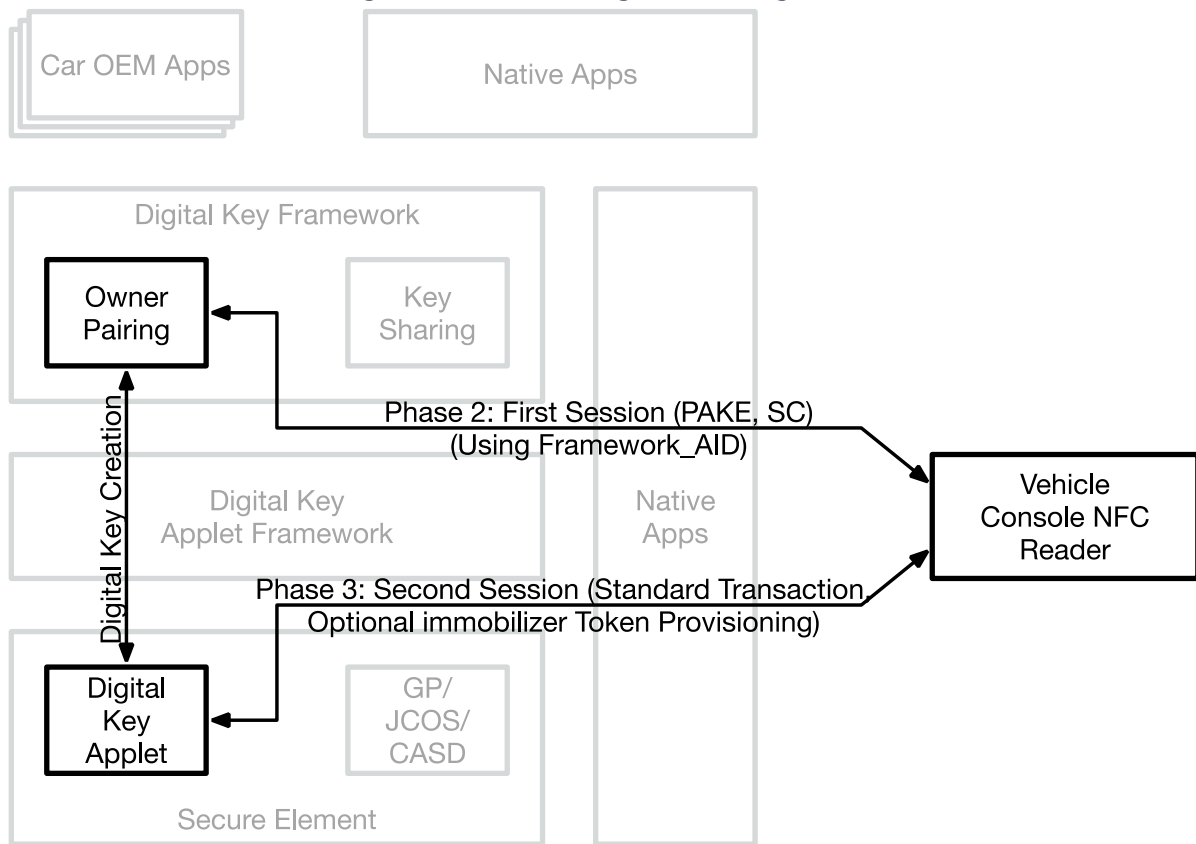
This section describes the phases of the owner pairing flow, which include:

- **Phase 0**: Preparation (see Section [6.3.1](#))
- **Phase 1**: Initiate pairing procedure on vehicle and device (see Section [6.3.2](#))
- **Phase 2**: First session with NFC reader (see Section [6.3.3](#))
- **Phase 3**: Second session with NFC reader (see Section [6.3.4](#))
- **Phase 4**: Finalization of pairing procedure (see Section [6.3.5](#))

The owner pairing flow consists of two sessions: the first session (Phase 2), executed with the Digital Key framework, and the second session (Phase 3) with the Digital Key applet, as shown in [Figure 6-1](#). In owner pairing Phase 2, the vehicle configures whether the owner device has to retrieve the immobilizer tokens online from the Vehicle OEM Server or from the vehicle. If online retrieval is configured, the owner immobilizer token is not transferred in the second session with the NFC reader but within the Key Tracking Request/Response to/from the Vehicle OEM Server over the owner Device OEM Server during or after the owner pairing. Figure 6-1 describes the owner pairing process whereby the immobilizer tokens are retrieved from the vehicle.

The first session consists of two NFC transactions. The first transaction is to negotiate protocol versions, execute the SPAKE2+, and transmit all key creation data to the device. The second transaction, which is executed after the creation of the Digital Key in the device, provides the creation attestation and certificate chain to the vehicle.

Figure 6-1: Owner Pairing NFC Exchanges



6.3.1 Phase 0: Preparation

6.3.1.1 Device Preparation

It is assumed that the device has already installed a Digital Key applet and created an Instance CA per Vehicle OEM prior to owner pairing operation. The Digital Key framework is updated with the list of Vehicle OEM partners.

The Instance CA Certificates are obtained by the Digital Key framework.

See [Listing 15-15](#) for the description of the Instance CA Certificate [E]. All signatures are generated as described in Section [18.4.10](#).

6.3.1.2 Vehicle Provisioning

The owner pairing secure channel creation is based on the SPAKE2+ protocol, which is described in Section [18.4.1](#). The protocol binds the device to the user and is resistant against eavesdropping and MITM attacks. See [\[10\]](#) for more information about the computational and security aspects of the protocol.

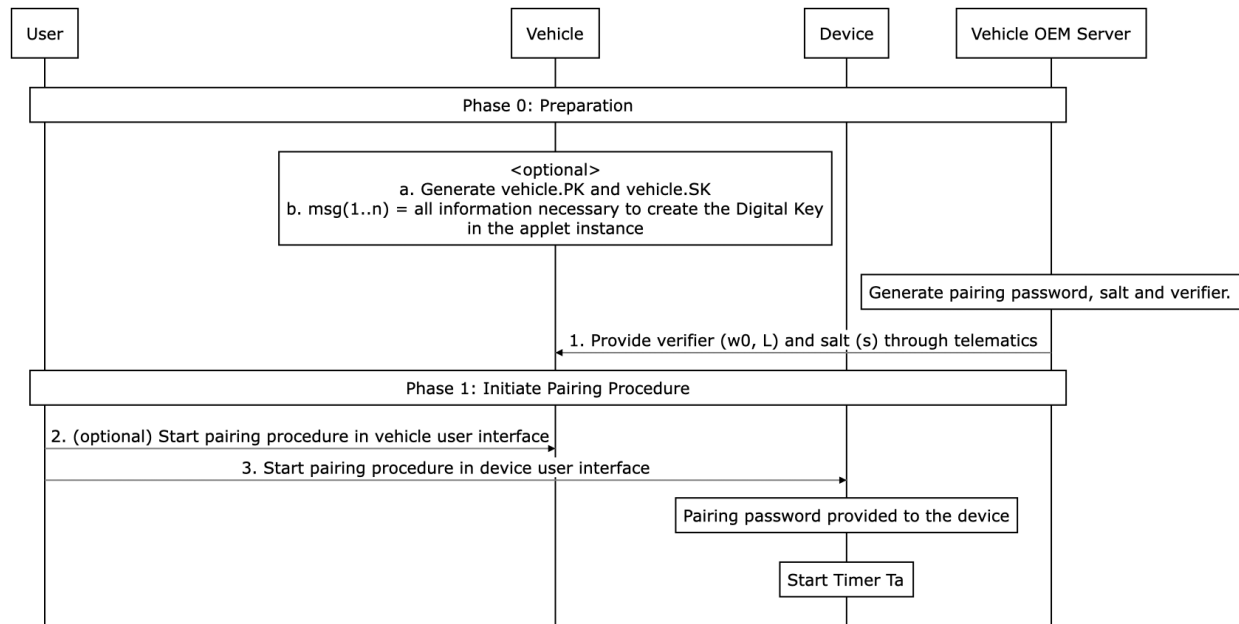
Before the owner pairing procedure starts, the Vehicle OEM Server generates the pairing password for the device and the password verifier for the vehicle as described in [Listing 18-1](#),

and sends the verifier and salt through a Vehicle OEM proprietary secure channel to the vehicle, as shown in Step 1 of [Figure 6-2](#). To enable SiaC feature the vehicle shall configure bits 4 and 5 in option_group_1 ([Table 15-13](#)) to 1.

6.3.2 Phase 1: Initiate Pairing Procedure

On the vehicle side, the pairing procedure initiation is under the responsibility of the Vehicle OEM. Appropriate owner pairing preconditions need to be met, e.g., the presence of a key fob.

Figure 6-2: Owner Pairing Flow-- Phase 0/1: Preparation/Initiation



The vehicle is either set into pairing mode by the user or tries to select the framework AID on the console NFC reader in the vehicle as long as no owner device is successfully paired.

To start owner pairing mode (Step 3 of [Figure 6-2](#)), the device receives the password, either through user input, a URL (see [Section 6.3.7](#)), or through an API directly from the Vehicle OEM app, if installed.

The device shall turn off the NFC interface after the pairing procedure has been started in the device (step 3 of [Figure 6-2](#)). The NFC link shall be started again after the user has entered the pairing password, or the user interface is dismissed.

NOTE: The deactivation of the NFC interface is introduced to stop additional polling due to external influence on the device or vehicle. For example, with activated NFC interface, while waiting for the input of the User (Step 3 of [Figure 6-2](#)), due to movements of the device the vehicle could restart the polling again. This could lead to a restart of the owner pairing while the first one is still running.

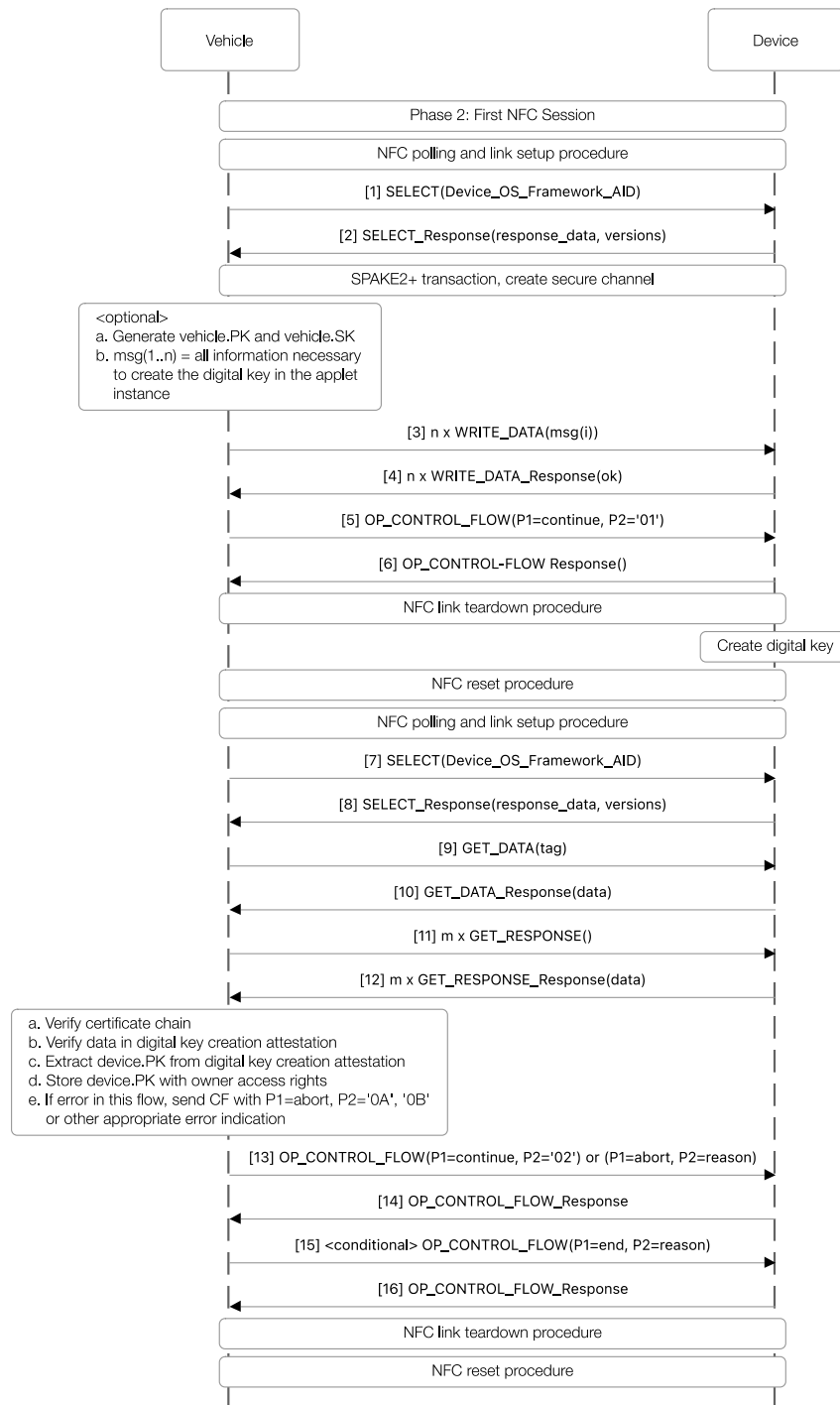
6.3.3 *Phase 2: First Session with NFC Reader*

The first NFC session consists of two distinct NFC transactions.

The first transaction negotiates the protocol versions (see Section [6.3.3.7](#)), mutually authenticates both device and vehicle, establishes a secure channel using SPAKE2+, and transmits all key creation data to the device, as shown in [Figure 6-3](#).

1

Figure 6-3: Owner Pairing Flow-- Phase 2: First NFC Session



2

3 The SPAKE2+ creates a symmetric session key pair used to establish a secure channel between
4 vehicle and device.

5 Before the second NFC transaction starts, the device creates the Device Key.

1 In the second NFC transaction, the vehicle reads the key creation data from the device, verifies it
2 and, if successful, stores the device public key. The NFC reset procedure is performed at the end
3 of each transaction.

4 The following steps or sequence of events takes place in the first NFC session:

5 *6.3.3.1 Step 1: Digital Key Framework Selection*

6 When the device is communicating to the console NFC reader in the vehicle, the vehicle shall
7 select the Digital Key framework using its AID, through the SELECT command. If the vehicle
8 selects the Digital Key Applet AID before the framework AID, then the device shall respond to
9 SELECT Digital Key Applet AID with Status Word (SW) = 6A82_h. The device shall return all
10 supported SPAKE2+ versions and all Digital Key applet protocol versions to the vehicle through
11 SELECT Response command. This determines the SPAKE2+ version (used for creating secure
12 channel) and Digital Key applet protocol version (for key sharing) to be used on both sides.

13 The version information shall be checked for compatibility on the vehicle side before continuing
14 as described in Section [6.3.3.7](#).

15 The SELECT command is described in Section [5.1.1](#).

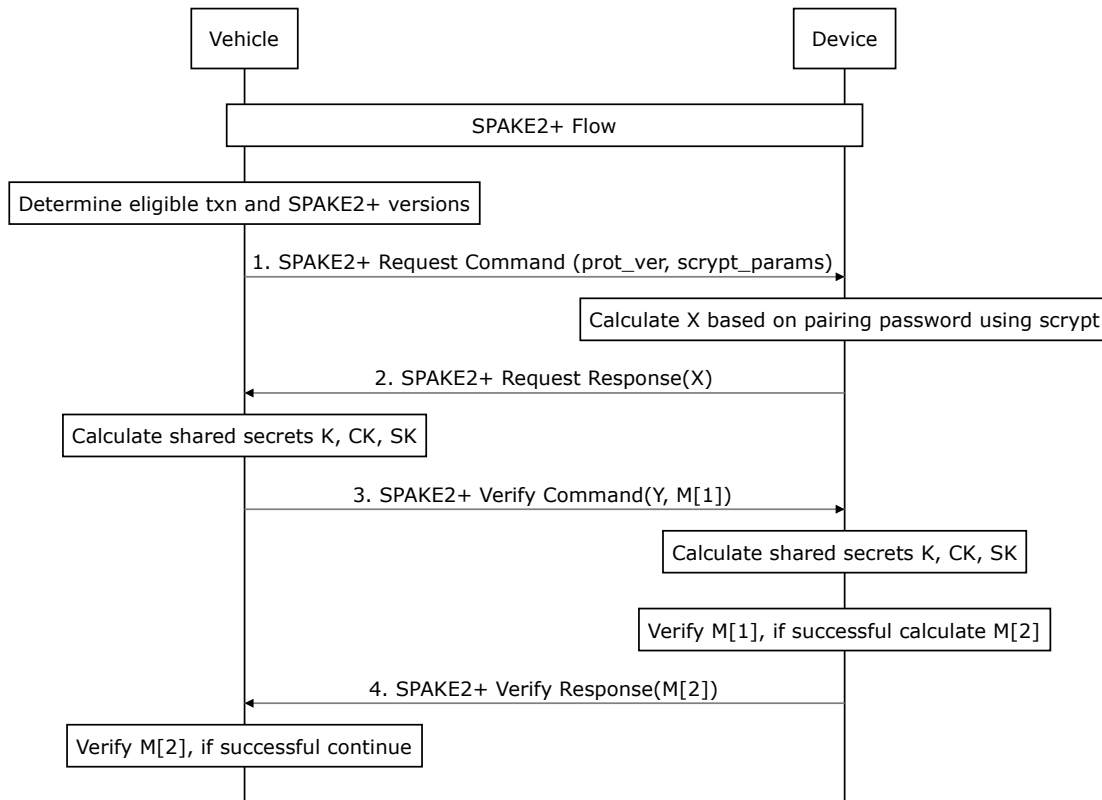
16 *6.3.3.2 Step 2 and 2a: SPAKE2+ Transaction*

17 The vehicle shall select and send to the device the SPAKE2+ protocol version to be used and the
18 list of all supported Digital Key applet protocol versions (as described in Section [6.3.3.7](#)).

19 The SPAKE2+ transaction is described in [\[10\]](#).

1

Figure 6-4: SPAKE2+ Flow



2

3 The SPAKE2+ transaction is shown in [Figure 6-4](#). The SPAKE2+ transaction establishes a
4 secure channel for the data exchange between vehicle and device. The secure channel shall
5 remain active across the RF resets.

6 The secure channel is terminated when the OP CONTROL FLOW command indicating success
7 is sent (Step 17 of [Figure 6-3](#) when an OP CONTROL FLOW command indicates an error, or
8 after any other abortion of the flow. An incorrect decryption or MAC value on APDU commands
9 also terminates the secure channel.

10 When the vehicle is not ready for owner pairing, it shall use the OP CONTROL FLOW to abort
11 and indicate the condition to the user, such as “preconditions for owner pairing not fulfilled.”

12 *Note:* Only commands with the appropriate class byte indication (bit 2 = 1) are part of the secure
13 channel.

14 6.3.3.3 Steps 3 to 4: WRITE DATA

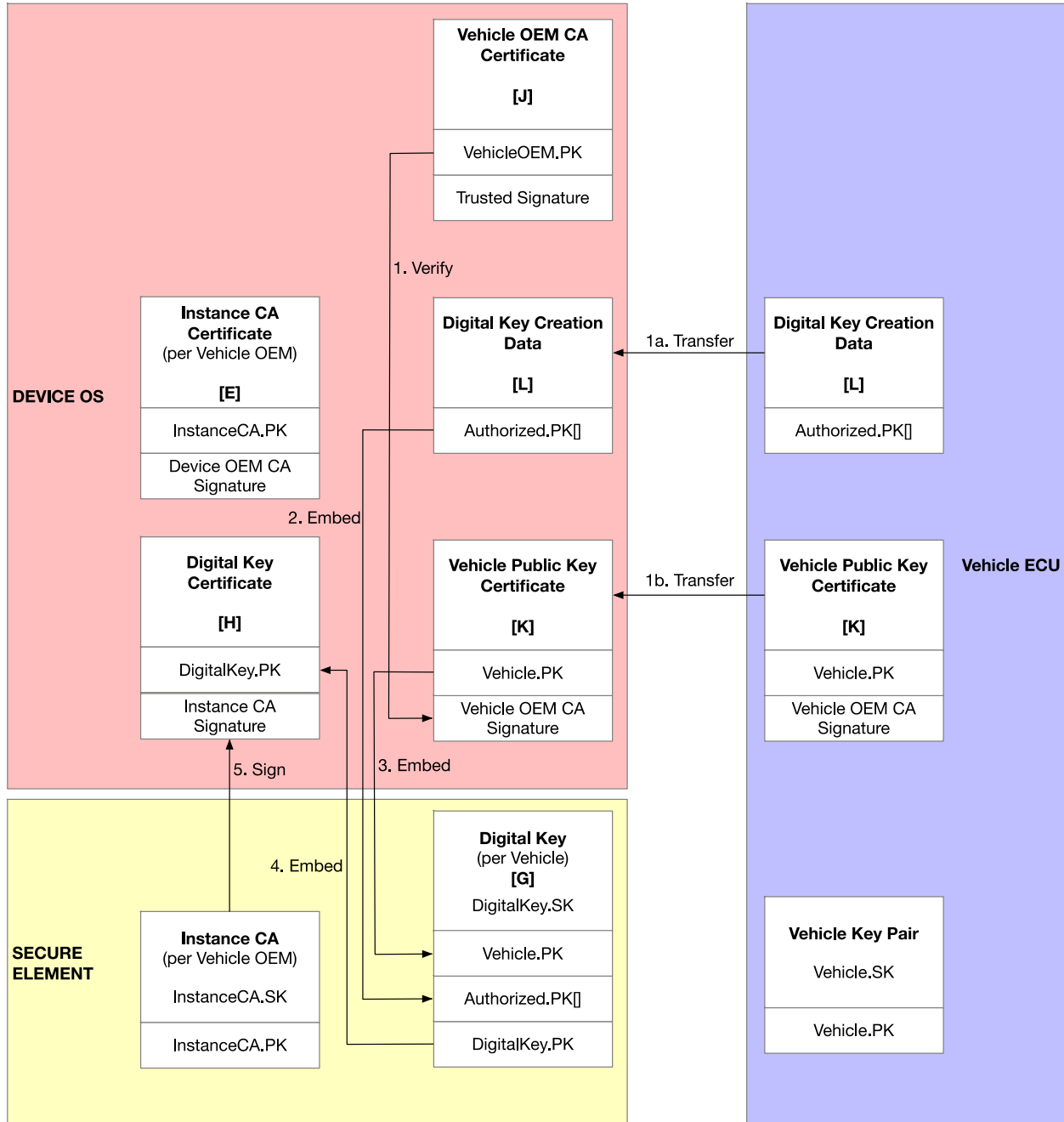
15 The vehicle shall send all data required to create a Digital Key through the secure channel to the
16 device using multiple WRITE DATA commands (see [Figure 6-5](#) and [Figure 6-6](#)).

17 After successful reception, the device shall verify the Vehicle Public Key Certificate [K] as
18 described in Section [6.3.3.9](#), using one of the following:

- 19 • Vehicle OEM CA Certificate [J] (see [Figure 6-5](#))
- 20 • Vehicle OEM CA Certificate [M] signed by Device OEM CA [D] (see [Figure 6-6](#))

- 1 In case of SE-centric applet model implementation, the Digital Key framework may omit the
- 2 verification since it is conducted by the Digital Key applet during Endpoint creation as described
- 3 in Section [15.3.2.4](#).

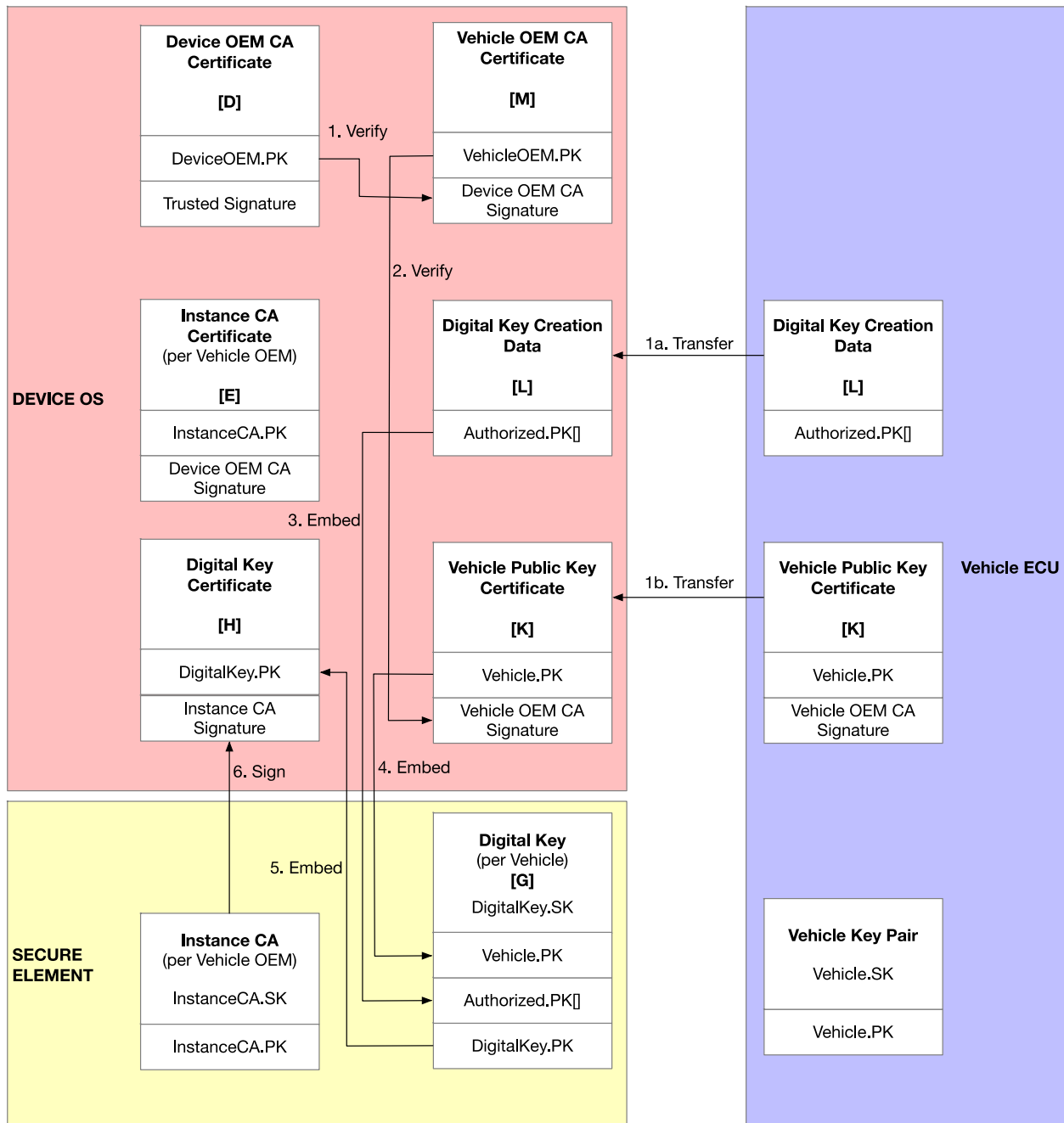
4 *Figure 6-5: Key Creation Data Transfer to Device*



5

1

Figure 6-6: Key Creation Data Transfer to Device



2

3 Certificate details are specified in Section [16](#).

4 **6.3.3.4 Steps 5 to 6: OP CONTROL FLOW**

5 The OP CONTROL FLOW command indicates that all data has been transmitted without error.

6 The device OS then switches off card emulation and creates the Digital Key as described in
7 section [6.3.3.8](#).

6.3.3.5 Step 7 to 12: *GETDATA*

The derived secure channel shall continue to be used in the second session.

Before sending the GET DATA command, the vehicle shall perform the NFC polling and link setup procedure and reselect the device OS framework AID using SELECT command.

The vehicle shall poll for the device to respond before considering the key creation as failed. See also section [6.3.3.11](#). For timeouts in case the device is removed from the reader or other connectivity issues occur, see Section [6.3.6](#).

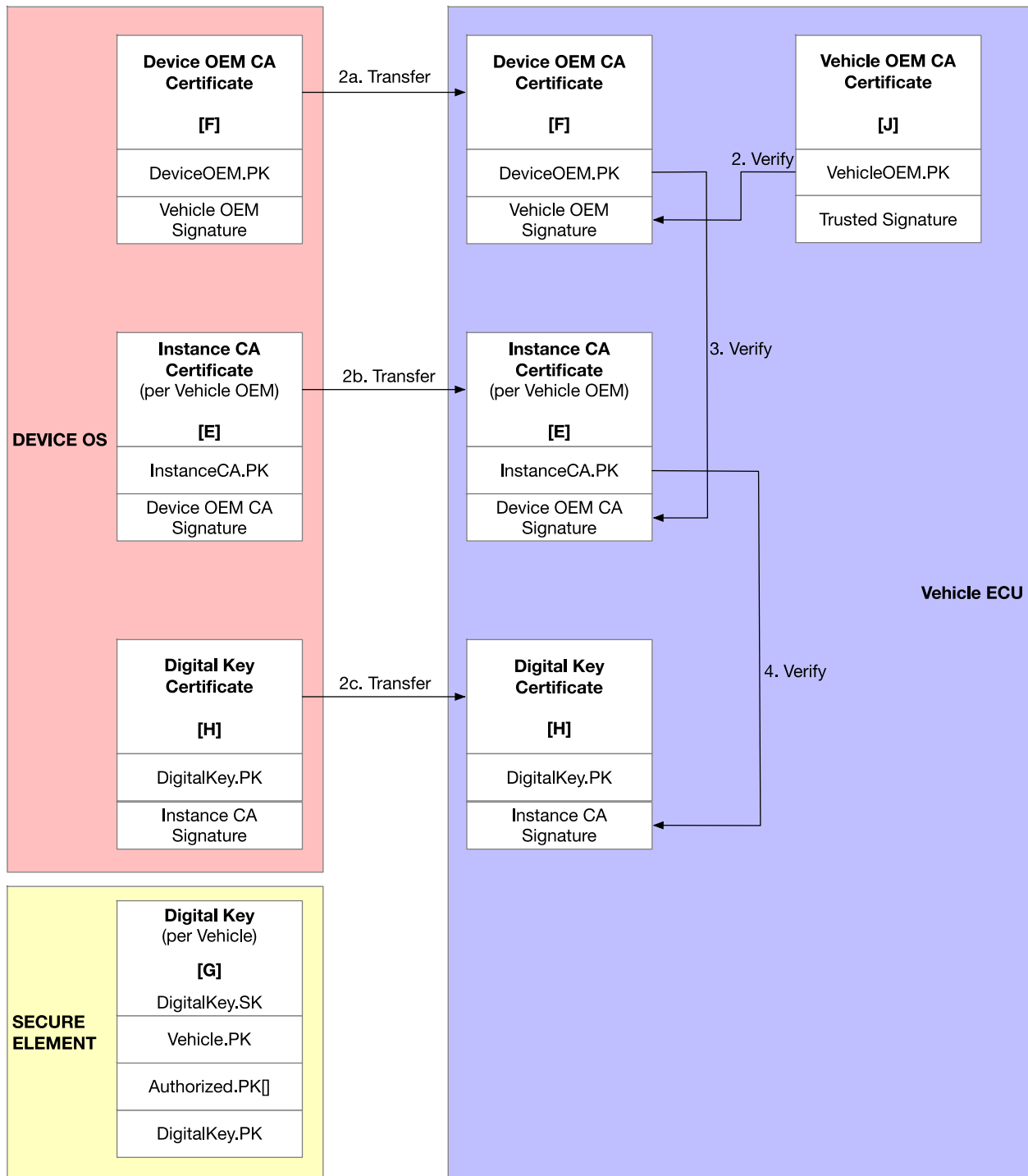
The vehicle shall send GET DATA command to request the Device OEM CA Certificate signed by the Vehicle OEM [F] (see [Listing 15-13](#)), the Instance CA Certificate signed by the Device OEM CA [E] ([Listing 15-16](#)), and the Digital Key Certificate from the Instance CA [H] (see [Listing 15-5](#)).

The vehicle shall verify all certificates and the relevant data elements as described in Section [6.3.3.10](#). If all the steps are successful, the vehicle shall store the device public key as the owner's public key.

The command GET DATA is described in Section [5.1.5](#).

1

Figure 6-7: Key Creation Info Retrieval by Vehicle



2

3 Certificate details are specified in Section [16](#).

6.3.3.6 Step 13 to 16: OP CONTROL FLOW

If vehicle has received all key certificates without error, then it shall send OP CONTROL FLOW (P1=10_h, P2=02_h) in step 13. Otherwise, the vehicle shall send OP CONTROL FLOW (P1=12_h, P2=reason) to abort the owner pairing due to error as indicated by the P2 value.

If the OP CONTROL FLOW in step 13 is with P1=10_h, P2=02_h, then the vehicle shall send OP CONTROL FLOW (P1=11_h, P2=11_h) in step 15 to end the owner pairing Phase 2 flow.

6.3.3.7 Checking the Protocol Version

Device and vehicle may support different versions of SPAKE2+ and Digital Key applet protocols. Device and vehicle shall support older but not deprecated protocol versions.

The owner device shall first return all the supported versions of SPAKE2+ and Digital Key applet protocols in the SELECT response. The vehicle shall return the highest matching version of the SPAKE2+ protocol and the agreed version of the Digital Key applet protocol at the beginning of the list, followed by all lower versions supported, ordered from the highest to the lowest. The list of all the supported Digital Key applet protocol versions allows the vehicle and owner device to find a different but compatible version on the receiver device for key sharing. An example is given below.

When no matching version can be determined, the owner pairing procedure shall be aborted. In this case, the vehicle should indicate this on its UI and shall send an error message to the device using the OP CONTROL FLOW command with “abort” indication.

Example of supported Digital Key applet protocol versions:

The Digital Key applet protocol versions supported by the vehicle (m supported Digital Key applet protocol versions (ver.high | ver.low)) as contained in SPAKE2+ REQUEST command are received by the device:

- 0300_h
- 0100_h

The Digital Key applet protocol versions supported by the device (m supported Digital Key applet protocol versions (ver.high | ver.low)) as contained in SELECT RESPONSE are received by the vehicle

- 0300_h
- 0100_h

The SELECT RESPONSE APDU would have a TLV with Tag 5C_h: 5C 04 03000100. The SPAKE2+ REQUEST command would have a TLV with Tag 5C_h: 5C 04 03000100.

6.3.3.8 Creating the Digital Key

The Digital Key framework creates a Digital Key in the SE using the CREATE ENDPOINT command (see Section 15.3.2.4) with the following parameters:

- **vehicle identifier:** Assigned by the vehicle. It is recommended to apply 2 bytes Vehicle Brand identifier (see [35]) and 6 bytes unique identifier (within Vehicle OEM) to the vehicle identifier. As the vehicle identifier is transmitted in the AUTH0 command, it shall be changed when the owner changes (i.e., when the vehicle goes through an “unpaired” state; see Section 2.7 for a description of pairing states) if needed for vehicle privacy.

- 1 • **endpoint identifier**: The common name in the subject identifier in the related certificate. It is
2 used to identify the Digital Key. The endpoint identifier is unique for each Digital Key in an
3 applet instance. The format is defined in [Appendix B.1](#).
- 4 • **Instance CA identifier**: Determines the Instance CA which signs the Digital Key Certificate
5 after key creation. The value shall match exactly the common name in the **subject identifier**
6 **of the Instance CA** to be associated (see [Listing 15-16](#)). The format is described in
7 [Appendix B](#).
- 8 • **Digital Key option group 1 and 2**: Should be set according to the Vehicle OEM policy.
- 9 • **protocol version**: The agreed upon Digital Key applet protocol version for the Digital Key.
- 10 • **vehicle public key**: The public key of the vehicle. For privacy reasons the key may change
11 when the owner changes (i.e., when the vehicle is going through the “unpaired” state; see
12 Section [2.7](#)).
- 13 • **authorized.PK[]**: This array shall contain the Vehicle OEM CA PK. Note that the authorized
14 public key shall not be updated once the Digital Key is created.
- 15 • **confidential mailbox size**: When immobilizer tokens are not needed, the size shall be set to
16 0. Otherwise, the size shall be (IMMOBILIZER_TOKEN_LENGTH) bytes (see [Table 4-4](#)).
- 17 • **private mailbox size**: Should be set according to the Digital Key structure and attestation
18 sizes, as defined by the Vehicle OEM and provided by the vehicle in the WRITE DATA
19 command (see Section [4.3.1](#)).
- 20 • **slot identifier**: Provided by the vehicle. It is used for Digital Key identification and anti-
21 replay of the deleted Digital Keys as described in Section [13](#). The vehicle shall provide slot
22 identifier for owner during owner pairing. Depending on vehicle OEM's implementation, the
23 slot identifiers for sharing (for friend) may be retrieved by the owner from vehicle or by the
24 receiver online from Vehicle OEM server.
- 25 • **counter limit**: Deprecated, shall not be provided.

26 An owner slot identifier shall in all cases (slot identifier retrieved from vehicle or online) be
27 provided by the vehicle in the endpoint creation data. The value provided here is used for the
28 standard transaction in Phase 3 of owner pairing.

29 If a Digital Key corresponding to the vehicle identifier already exists in the device, the
30 framework shall terminate the existing Digital Key and send termination attestation along with
31 the Vehicle OEM proprietary data to KTS (if applicable) before creating a new Digital Key.

32 6.3.3.9 Verifying the Vehicle Public Key Certificate Chain

33 The device shall parse the certificate chain listed below and validate the critical data elements
34 and the signature before accepting the vehicle public key:

- 35 • Vehicle Public Key Certificate format [K]
- 36 • Intermediate Certificate (optional)
- 37 • Vehicle OEM CA Certificate [J] or Device OEM CA signed Vehicle OEM CA Certificate
38 [M]

39 All certificates shall be in X.509/ASN.1 format and shall be DER encoded.

The formats of the subject identifier and issuer identifier are defined in [Appendix B.1](#). After successful verification of the vehicle public Key Certificate Chain, the owner pairing Phase 2 is completed.

6.3.3.10 Verifying the Endpoint Creation Attestation Chain

The vehicle shall parse the certificate chain listed below and validate all data elements, following the rules defined in Section [15](#), before accepting the device public key:

- Device OEM CA Certificate (signed by Vehicle OEM) [F]
- Instance CA Certificate (signed by Device OEM) [E]
- Digital Key Certificate [H]

All certificates shall be in X.509/ASN.1 format and shall be DER encoded.

6.3.3.11 Error Handling for the first NFC session

The following errors may appear in the first session of the NFC pairing flow:

- General errors
 - Communication error
 - RF transmission error
 - owner removes device before end of protocol flow, i.e., no deselect command received before link loss
 - commands not received in correct order
 - unknown command received
 - device sends other error status word
 - implementation bug leading to timeout
- SELECT command failure
 - owner pairing not activated on device
- SPAKE2+ protocol failure
 - wrong password entered on device
 - maximum number of pairing attempts reached (enforced by the vehicle)
 - protocol version mismatch
 - vehicle sends OP CONTROL FLOW command with error indication
- Owner key creation on device side fails
 - Vehicle Public Key Certificate [K] verification failed
 - Digital Key Creation Data inconsistency
 - Digital Key configuration not allowed
 - not enough memory in applet to create the Digital Key

The device shall enforce that the vehicle sends the commands in the specified order. If this does not occur, the device shall abort the transaction with the error code “Command used out of sequence.”

Vehicle and device shall allow up to a maximum of seven retries (using try_cnt, try counter) of the first session. If the maximum number of cumulated failures is reached, vehicle or device shall end the pairing mode and re-provision a new SPAKE2+ verifier.

When no matching protocol versions for SPAKE2+ or Digital Key transaction protocol can be found by the vehicle, it shall send an appropriate OP CONTROL FLOW command (namely, “abort”) and shall not send the SPAKE2+ REQUEST command. In this case the device does not expect additional commands from the vehicle.

After the OP CONTROL FLOW command, the vehicle shall perform the NFC link teardown procedure followed by the NFC reset procedure, before a new or updated device is ready to begin the first step in owner pairing (Phase 2) again.

In the case of a failed transaction due to tearing that cannot be compensated by the contactless protocol, the vehicle shall perform the NFC reset procedure (as defined in Section 3.2.4) and start the NFC polling and link setup procedure.

In case of an application timeout (not on contactless protocol level), the vehicle shall send an OP CONTROL FLOW command indicating a timeout error.

If any error occurs during the first NFC session before key creation, no Digital Key shall be created in the Digital Key applet.

When a communication error occurs, the Digital Key framework shall not delete the Digital Key and shall wait for the vehicle to retry the flow.

If a device sends an error SW (i.e., SW not equal 9000_h or 61XX_h) which is listed in the table below, the listed behavior shall occur.

Table 6-1: Error SW Condition List 1.

Command	Error SW	Device Action	Vehicle Action
SPAKE2+ VERIFY	6A88 _h	Stop HCE UI: “Wrong Password”	Reset

When a unlisted error SW is sent, the error counter shall be increased, and the flow shall be retried if the counter has not yet reached the maximum.

When an error is indicated in an OP CONTROL FLOW command after the Digital Key is created and the vehicle rejects the pairing due to failing verification of the Digital Key Certificate data, the Digital Key framework shall delete the Digital Key that has been created.

On receipt of an OP CONTROL FLOW command indicating a timeout error, if the maximum number of cumulated failures is reached, vehicle or device shall end the pairing mode. The user shall request a restart of the owner pairing procedure for which the Vehicle OEM Server has to re-provision a new SPAKE2+ verifier and password.

When the device detects a link loss or receives an unexpected command without receiving an OP CONTROL FLOW command immediately beforehand, the transaction shall be considered as failed and should be allowed to be repeated a limited number of times (e.g., 5).

When the Digital Key has been created but the Digital Key Certificate [H] has not been requested by the vehicle, the Digital Key shall not be usable.

A policy on the vehicle side shall limit the number of unsuccessful owner pairing attempts per set of pairing password/vehicle verifier to seven as outlined in Section 5.1.2. When a successful pairing is completed or when the maximum number of failed pairing attempts is reached, the

vehicle shall delete the verifier immediately. In case of a successful owner pairing, the verifier shall not be updated before the vehicle has verified the key tracking signature unless it was invalidated for a different reason.

When the owner pairing flow is permanently aborted (e.g., via timeout on device, maximum number of retries reached, etc.), the created Digital Key shall be deleted by the device.

During owner pairing Phase 2, the device shall prevent selection of the Digital Key applet by the vehicle and shall respond to SELECT Digital Key applet AID with Status Word (SW) = 6A82.

6.3.4 Phase 3: Second Session with NFC Reader

The device shall allow selection of the Digital Key Applet by the vehicle. The second NFC session is executed between vehicle and Digital Key applet as described in [Figure 6-8](#).

6.3.4.1 Steps 3 to 12: Standard Transaction

The vehicle shall select the Digital Key applet and execute a standard transaction (step 3–6; see [Section 7](#)) using transaction type 07_h (see [Table 9-1](#)).

In response to the SELECT Digital Key applet AID over NFC, the vehicle may receive an Error Status Word (SW) equal to 6A82_h indicating that the selection of the Digital Key applet is not yet available. Upon reception of this status word, the vehicle should retry the selection of the Digital Key applet every $T_{veh-selectRetry}$ until the expiration of TO_{veh-4} .

The vehicle shall first write the opaque attestation into the private mailbox. If the immobilizer token is required and retrieved from vehicle, it shall be written into the confidential mailbox of the owner Digital Key in the Digital Key applet.

In order to write the appropriate data structures into the confidential and private mailboxes, the vehicle shall use the EXCHANGE command (see [Section 15.3.2.15](#)) with the appropriate parameters to:

- Write the opaque attestation with the tag 5F5A_h tag ([Table 6-2](#)) into the KeyAtt field in the private mailbox. 5F5F_h shall be the last TLV written to mark the end of the transmitted data sequence.
- Write the owner immobilizer token into the confidential mailbox (if immobilizer tokens are retrieved from the vehicle)
- Write the owner slot identifier into the SlotIdentLookup in the private mailbox (conditional)
- Write the Vehicle OEM proprietary data structure into the VehiclePropDat in the private mailbox using VEHICLE_KEY_ATT_BIT and VEHICLE_OEM_PROPRIETARY_DATA_FOR_DEVICE_BIT.
- Write the signaling bitmap (SigBmp) as per [Table 4-2](#) to indicate the update of the above data structures into the private mailbox

Table 6-2: Objects for Device Digital Key Certificate

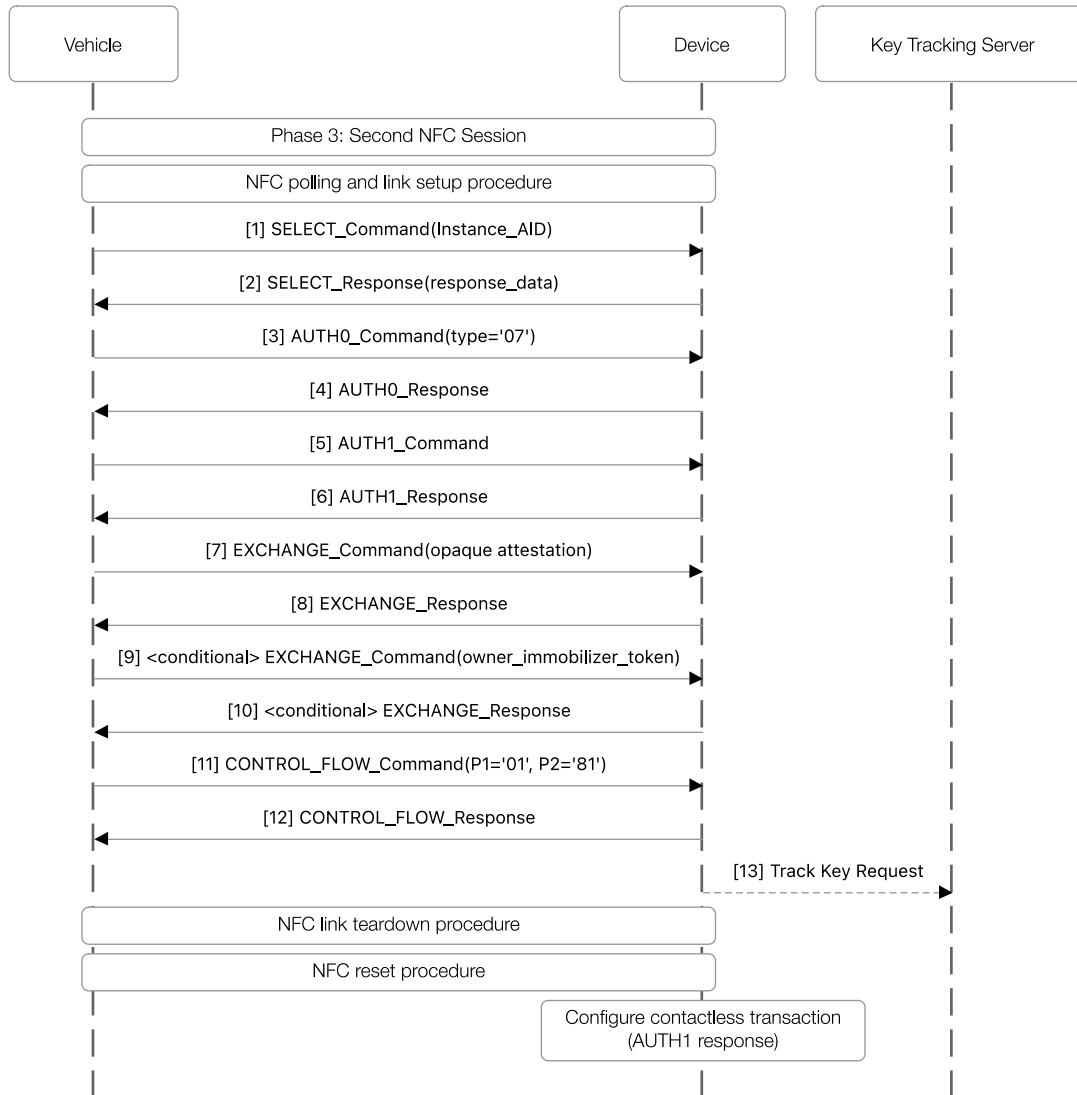
Tag	Length (bytes)	Data Content	Field is
5F5A _h	variable	Attestation of the device PK by the vehicle (opaque)	optional
5F5F _h	0	Completion of Digital Key Certificate Sending	mandatory

The owner slot identifier is not required as it is included in the endpoint certificate. The vehicle OEM decides whether it needs to be written or not.

It is assumed that no owner access rights need to be provisioned into the Digital Key applet. The vehicle shall grant full access to the owner.

The Digital Key applet shall notify the framework at the end of the transaction, which then shall configure the contactless transaction as described in Section [6.3.4.3](#).

Figure 6-8: Owner Pairing Flow – Phase 3: Second NFC Session



6.3.4.2 Steps 11 and 12: CONTROL FLOW

The vehicle shall send the CONTROL FLOW command (see Section [15.3.2.16](#)) to indicate that all data has been written successfully into the mailboxes. The P2 parameters shall be set as follows:

P2= 81_h to indicate to the Digital Key applet that vehicle has completed owner pairing phase 3.

At the end of Phase 3, the owner immobilizer token shall be provisioned in the owner device if immobilizer token is retrieved from the vehicle, and the key may or may not yet be tracked by the vehicle.

6.3.4.3 Step 13: Key Tracking Request

If the standard transaction was successful, the owner device shall send an asynchronous message to register the owner key to the KTS. Key registration is optional for the vehicle. If the vehicle attempts to register the key as well, a race condition may occur. Thus, key registration by vehicle is not recommended.

Note: When the standard transaction fails, the device shall not send a key tracking request to the server. The data elements listed in [Table 6-3](#) shall be sent to the KTS by the device.

The Account Info Hash is an anonymous identifier for the device OEM account created from the Device OEM account identifier (accountID) and the vehicle identifier. It shall be the same for the Digital Keys on devices on the same device OEM account and shall be different for Digital Keys on devices on different device OEM accounts. It shall be diversified per vehicle.

Account Info Hash = SHA-256 (accountID || vehicle_identifier || salt for vehicle OEM). The salt shall be different for every vehicle OEM.

Table 6-3: Owner Key Tracking Request

Tag	Length (bytes)	Description	Element is
7F3E _h	variable	Owner key tracking request	
		[E] Instance CA Certificate per Vehicle OEM. See Figure 6-7 and Table 11-8 .	mandatory
		[H] Digital Key Certificate. See Figure 6-7 and Table 11-9 .	mandatory
		Endpoint encryption key attestation signed by endpoint private key. See Table 15-49	Conditional. Present only if immobilizer token is required
D5 _h	variable	Attestation of the device PK by the vehicle (opaque)	mandatory
D3 _h	variable	Friendly name of the owner key	mandatory
5E _h	32	Account Info Hash	mandatory
5F49 _h	65	Device privacy encryption key (Device.Enc.PK)	mandatory
DA _h	Variable (maxlen: 10 bytes)	Device privacy encryption version Default: "ECIES_v1" (ASCII)	mandatory
DB _h	1	Owner Device Instance CA freshness requirement for Key Sharing in hours. No freshness check required if tag is absent or 0.	optional

Tag	Length (bytes)	Description	Element is
5B _h	2xn	V-OD-FW-vehicleList (ver.high ver.low), agreed version first as obtained in SPAKE2+ Request (see 5.1.2) during owner pairing.	

1 *Table 6-4: Owner Key Tracking Response Parameters*

Parameter	Length	Description
ktsSignature	variable	Delivered in data structure keyData Table 17-40
slotIdentifier	1-8	Delivered in data structure keyData Table 17-40
confidentialMailboxData	variable	Delivered in data structure keyData Table 17-40
kBleOobKey	variable	Delivered in data structure keyData Table 17-40
kBleIntroKey	variable	Delivered in data structure keyData Table 17-40
groupIdentifier	2	See Table 17-40
deviceType	1	See Table 17-57

2 All parameters of the Key Tracking Response from the KTS shall not be provided in TLV
3 format. The server API parameters shall be provided within the JSON structure defined in the
4 relevant sections referenced in [Table 6-4](#).

5 If the slot_identifier is provided, it shall be cryptographically bound to the ktsSignature (by
6 vehicle OEM proprietary methods not defined in this specification) so that the vehicle can verify
7 the authenticity of the slot_identifier.

8 The content of the ktsSignature is defined by the Vehicle OEM; it is opaque to the device OEM.
9 The signature shall have the following properties:

- 10 1. Contain cryptographic confirmation that the owner key has been tracked by the KTS and
11 shall link the signature to the device public key.
- 12 2. Cryptographic strength of the signature used to generate the ktsSignature in the
13 key_tracking_receipt shall match or exceed the cryptographic strength of the signature
14 used by the device OEM.

15 The length of the KTS Signature shall not exceed 320 bytes.

16 The device shall add the tag 45_h and length fields to the ktsSignature (see [17.11.1.6](#)) and stores the
17 TLV structure in the KeyAtt field of the private mailbox (see [Table 4-3](#)).

18 Endpoint encryption key attestation (Tag 7F26_h) shall be included in [Table 6-3](#) when Tag DA_h in
19 Tag 7F60_h in [Table 5-14](#) is set to 01_h.

20 Owner slot identifier shall be included in the Key Tracking Response as defined in [Table 6-4](#)
21 when Tag DA_h in Tag 7F60_h in [Table 5-14](#) is set to 01_h or 03_h.

22 Owner immobilizer elements (confidentialMailboxData and ephemeralPublicKey parameters)
23 shall be included in the Key Tracking Response as defined in [Table 6-4](#) when tag DA_h in tag
24 7F60_h in [Table 5-14](#) is set to 01_h.

25 The groupIdentifier is assigned by the vehicle OEM and is unique per vehicle and
26 AccountInfoHash. The device shall overwrite the kble_intro received in the Tag 7F49_h during
27 owner pairing with kBleIntroKey and kBleOobKey, if kBleIntroKey and kBleOobKey are

present in the owner key tracking response (see [Table 6-4](#)). The device shall use kBle_intro and kBleOobKey during First Approach of the owner device (see section [19.3.4](#)).

If the owner Key Tracking Response contains a slot identifier, then this slot identifier shall be written to the owner endpoint using the SETUP ENDPOINT command. If no slot identifier is provided in the owner key tracking response, then the slot identifier provided by the vehicle in the endpoint creation data is the one to be used.

All certificates in [Table 6-3](#) shall be in X.509/ASN.1 format and shall be DER encoded.

During key tracking for shared keys, Owner/Sender Device Instance CA freshness and validity check in vehicle OEM server is required if tag DB_h was present in owner/sender key tracking (see [Table 6-3](#)) and did not have the value 0.

The opaque attestation is obtained from the KeyAtt field of the private mailbox. Note that sending the attestation as part of the standard transaction is mandatory. Only value field of Tag 5F5A_h shall be included in the value field of Tag D5_h.

See Section [17.7.3](#) for details on the format of the key tracking request.

If immobilizer tokens and/or slot identifiers are retrieved online, the owner device shall send Key Tracking Request to the server (see Section [17.7.3](#)) to request owner immobilizer token and/or slot identifier.

6.3.4.4 Configuration of the Contactless Transaction

There are two options to access mailbox data within the standard transaction:

- Transmit the required data as part of the AUTH1 response
- Use the EXCHANGE command (mandatory for the fast transaction)

If the WRITE DATA command received from the vehicle contains tag 4A_h and/or tag 4B_h in the Device Configuration Data field (see step 3 in [Figure 6-3](#) and Section [5.1.4](#)), the device shall transmit the mailbox version, signaling bitmap, slotIdentLookup and/or owner immobilizer token in the AUTH1 response as described in [Table 15-35](#) and the Digital Key framework shall set the endpoint using the SETUP ENDPOINT command (see Section [15.3.2.21](#)) with the following parameters:

- **confidential offset:** Shall be set to 0 as defined in [Table 4-4](#).
- **confidential length:** Shall be set to IMMOBILIZER_TOKEN_LENGTH as defined in [Table 4-4](#) if immobilizer tokens are used.
- **private offset:** Shall be set to MBX_VER_OFFSET
- **private length:** Shall be set to SLOT_ID_OFFSET or SLOT_ID_OFFSET + length of SlotIdentLookup structure

The other arguments in this method shall not be used.

6.3.4.5 Error Handling for the second NFC session

The following errors that appear in the second session of the NFC pairing flow include:

- General errors that may include
 - Communication error
 - RF transmission error

- owner removes device before end of protocol flow, i.e., no deselect command received before link loss
- commands not received in correct order
- device sends other error status word
- implementation bug leading to timeout
- SELECT command failure
 - applet not installed
 - device configured for applet when HCE selected over NFC

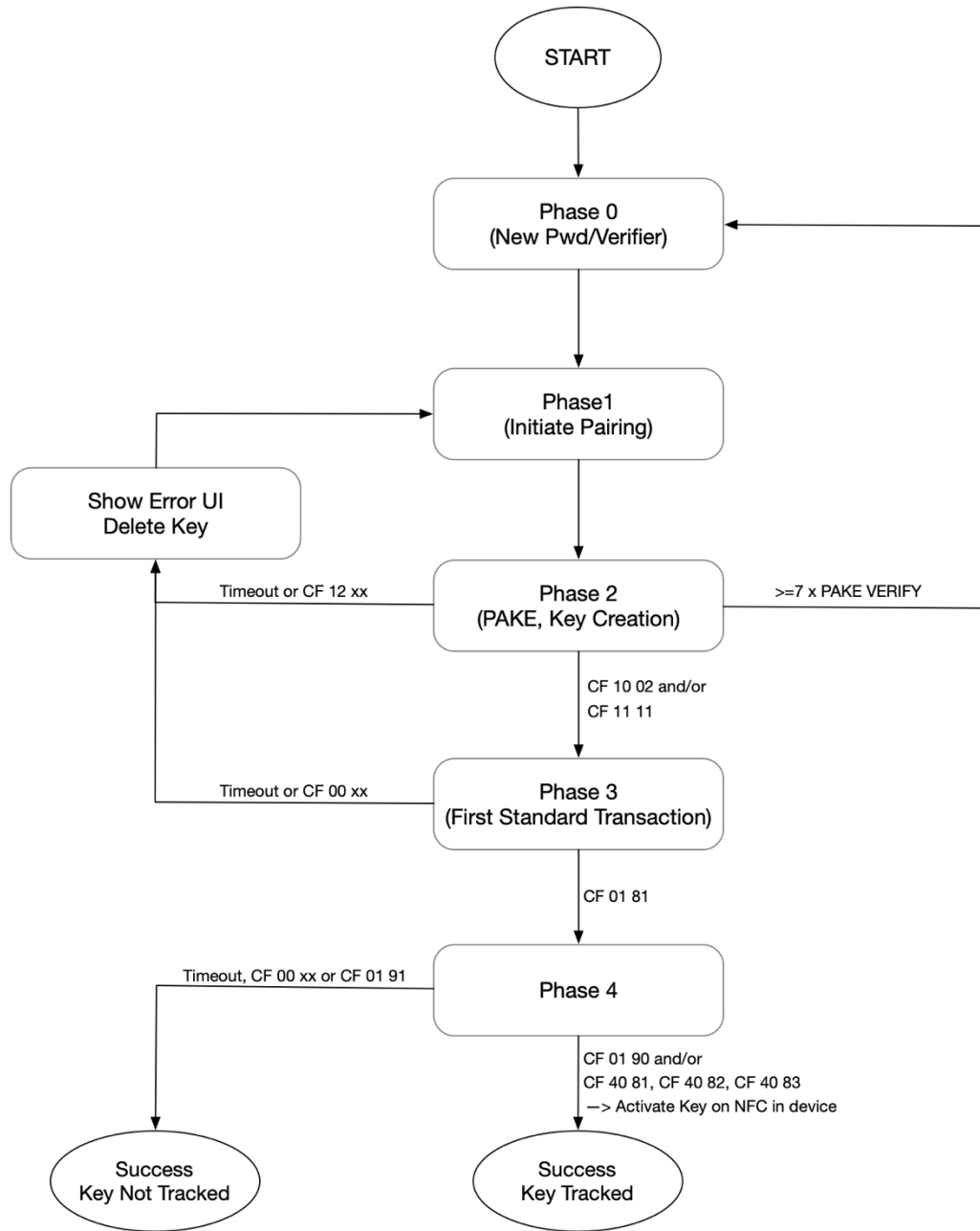
[Figure 6-9](#) shows the error management in owner pairing and the relationship between the different phases. No retry shall be allowed in a specific phase and any error which is not fixed on NFC protocol level shall lead to complete restart.

If the standard transaction (in step 3–6) failed, the endpoint may be deleted.

As soon as the Digital Key is successfully created, it might be usable on the contactless interface, even if the owner immobilizer token is not yet provisioned. For a consistent user experience, it is recommended to enable the created owner key only when the device has confirmed that the owner immobilizer token and the opaque attestation have been correctly written.

1

Figure 6-9: Error Management of Different Phases in Owner Pairing



2

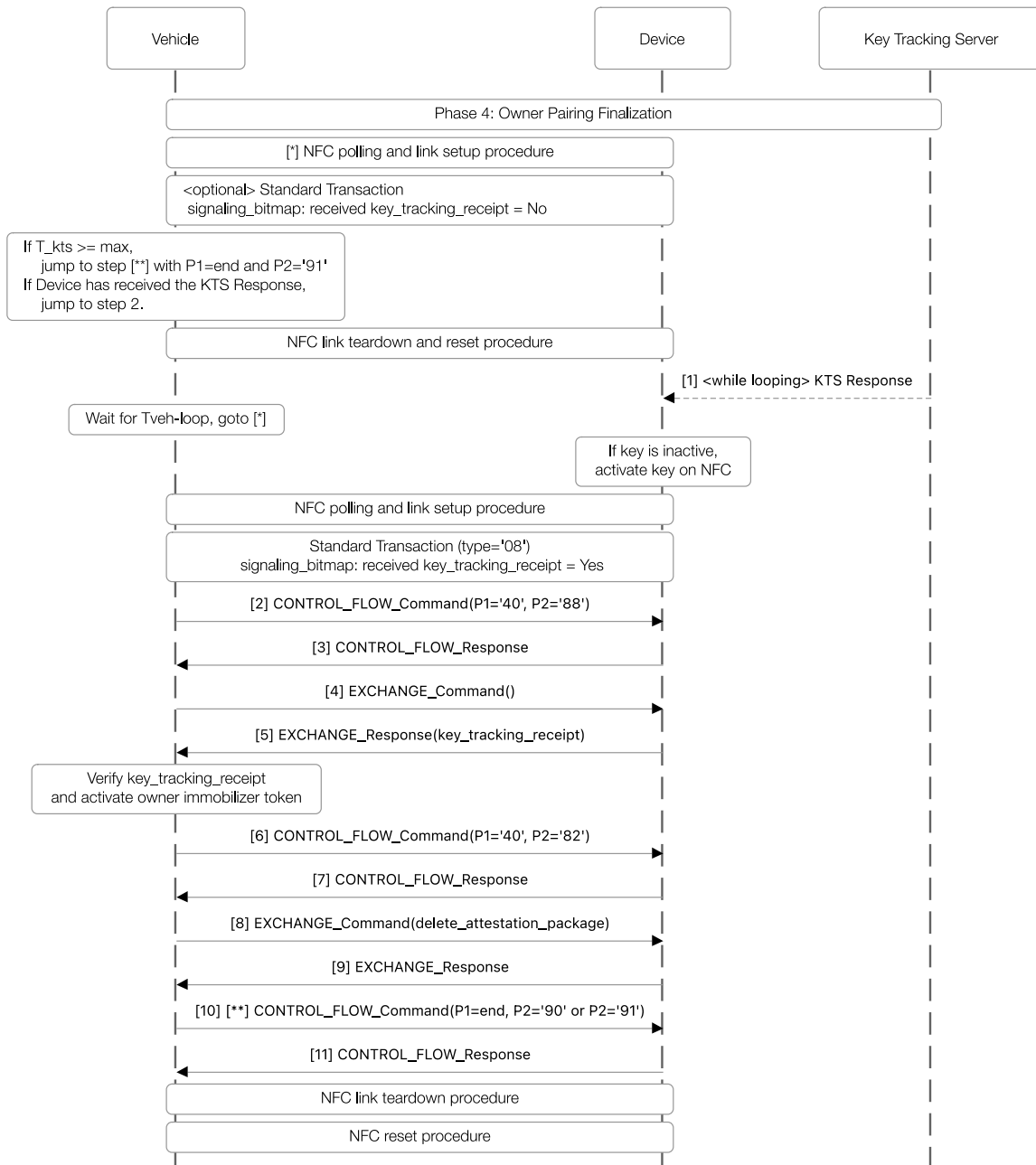
3 6.3.5 Phase 4: Finalization of Pairing Procedure

4 The finalization step executes a standard transaction using transaction type 08_h (see [Table 9-1](#)) in
5 regular intervals to poll the signaling bitmap indicator for an attestation package in the device.

6 [Figure 6-10](#) shows the final phase of the owner pairing flow.

1

Figure 6-10: Owner Pairing Flow – Phase 4: Finalization



2

3 6.3.5.1 Before Step 2: Reader Polls for KTS Response

4 The owner device and (optionally) the vehicle have reached out to the KTS by sending a key
5 tracking request, as defined in [Table 6-1](#) in Phase 2.

6 If the device receives a slot_identifier and Key Tracking Response from the KTS (see [Table 6-4](#)),
7 the device shall store the slot_identifier and ktsSignature in the SlotIdentLookup and KeyAtt
8 field in the private mailbox (see [Figure 4-4](#)), respectively. The corresponding bit in the signaling
9 bitmap (SigBmp) shall be set by the device to indicate to the vehicle the presence of the
10 slot_identifier and ktsSignature.

The reader executes standard transactions to check the signaling bitmap in the device. If the ktsSignature is not present or the device does not respond⁵, the reader shall try again after a time ($T_{veh-loop}$) defined in Section 6.3.6. The reader shall perform the NFC Reset procedure after each standard transaction.

If the reader experiences a loss of connection, it shall perform the NFC Reset procedure and shall restart the NFC Polling and Link Setup procedures.

If the vehicle receives the Key Tracking Response, the vehicle shall first skip the verification of the ktsSignature in the device and then continue by writing the receiver immobilizer tokens into the mailbox if immobilizer tokens are retrieved from the vehicle.

If neither vehicle nor device receives a response before $tO_{veh-kts}$ has expired, the vehicle shall not provision receiver immobilizer tokens and shall signal the failure to get a KTS signature through a CONTROL FLOW command (see Section 6.3.5.4).

Note: The device may not respond to NFC polling until it has received the key tracking response. The vehicle shall time out (by $tO_{veh-kts}$) even if the device never responds.

6.3.5.2 Steps 2 to 5: Verification of the Key Tracking Receipt in Device

When the device obtains a KTS signature, the vehicle shall execute a standard transaction and shall send a CONTROL FLOW command (see Section 15.3.2.16) to indicate the status:

1. P1=40h and P2=88h: continue, key tracking response received in device; next step is to read the receipt from the mailbox
2. P1=40h and P2=89h: continue, key tracking response received in vehicle, go directly to verification of key tracking receipt

Steps 2 through 9 shall be executed only when the KTS signature is stored in the device. If a key tracking response is received by the vehicle, only steps 2 and 3 are executed as defined in 2, above.

The vehicle shall then verify the KTS signature.

If the verification of the KTS signatures was successful, the vehicle shall continue with the next step. Otherwise, the vehicle shall abort and execute steps 14 and 15 with an appropriate error indication (see Section 6.3.5.4). If immobilizer tokens are not required, the vehicle can proceed directly to step 10 (see Section 6.3.5.3). Otherwise, the vehicle shall abort and execute steps 14 and 15 with an appropriate error indication.

6.3.5.3 Steps 6 to 9: Deleting the attestation package

If there is a key tracking receipt present in the private mailbox, which was already verified by the vehicle, the vehicle shall delete the key tracking receipt as follows:

The vehicle shall indicate the deletion of the key tracking receipt by sending a CONTROL FLOW with:

- P1=40_h and P2=82_h: continue, attestation package delete start.

⁵ A device may choose not to enable the key on the NFC interface or BLE/UWB interface until the ktsSignature is present in the private mailbox.

The vehicle shall clear the signaling bitmap and delete the attestation package from the private mailbox using an EXCHANGE command, so that the vehicle does not detect the same attestation package a second time, on the next transaction.

6.3.5.4 Steps 14 to 15: CONTROL FLOW

The CONTROL FLOW command indicates the end of the owner pairing flow with the following options:

1. P1=01_h and P2=90_h: end, owner key is tracked, and all data has been written successfully into the mailboxes.
2. P1=01_h and P2=91_h: end, owner key is not tracked, key sharing is not possible, and the owner needs to go online to track the key before using it.

6.3.6 Timers

The owner pairing flow is limited by retry counters and by timers on vehicle and device sides.

The vehicle limits the overall time of a pairing attempt, from the first detection of the device responding successfully to the selection of the framework AID until the successful provisioning of the immobilizer token in the Digital Key.

Phases and sessions are limited by shorter timer values, to be able to detect failures more quickly.

Vehicle overall timer $t_{O_{veh-1}}$ is used to control the owner pairing procedure. This timer is not linked to the device timer $t_{O_{dev-1}}$.

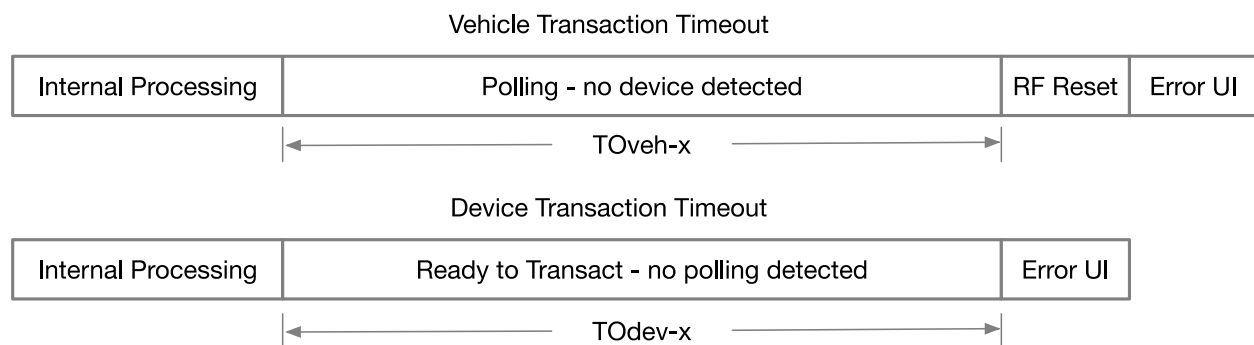
The device should at least have a timer ($t_{O_{dev-1}}$) to limit the overall time of the owner pairing procedure until the key and immobilizer token are provisioned. On timeout, owner pairing shall be considered to be successful (See [Figure 6-9](#)), and the previously created key shall be saved. The device should continue to track the key until a successful trackKey() Response is received.

If $t_{O_{veh-kts}}$ expires the vehicle shall signal success if phase 3 was successful and shall expect the KTS receipt to be presented on the next transaction, similar to first new key transaction.

A timer should not be restarted if it is already running, and the flow passes again through the “Start Timer” state (which is unlikely).

The following scheme provides more details about the timer start and end conditions. It shows the specific timeouts from vehicle and device perspective.

Figure 6-11: Vehicle and Device Transaction Timeout



- 1 [Table 6-5](#) defines the timer values. The timer numbers refer to [Figure 6-11](#) with a split into
2 vehicle and device side.

3 *Table 6-5: Recommended Minimum Vehicle and Device Timeout Values*

Timer	Description	Value
tO _{veh-1}	Overall OP transaction timer (detection of device until final UI)	30 sec
tO _{dev-1}	Overall OP transaction timer (first response to polling until KTS signature not received)	30 sec
tO _{veh-2}	not required	
tO _{dev-2}	Start of pairing mode until detection of polling	Device OEM specific
tO _{veh-3}	Last WRITE DATA command until first response to polling for GET DATA	10 sec
tO _{dev-3}	Endpoint created until detection of polling	5 sec
tO _{veh-4}	Endpoint data processed until first successful response to SELECT for first standard transaction	5 sec
tO _{dev-4}	Endpoint data sent to vehicle until detection of polling	10 sec
tO _{veh-5}	When performing owner pairing or first key transaction over BLE, maximum time that vehicle waits for between owner pairing or first key transaction phases to receive a Request_Standard_Transaction SubEvent. See Section 19.5.8	2 sec
T _{veh-loop}	Time of repeated standard transaction to retrieve Key tracking receipt (Note 1)-- this is not a timeout, but a loop timer	1 sec
tO _{veh-kts}	Time until end of Owner Pairing Phase 4 (success if key is created but not tracked).	20 sec
T _{veh-SelectRetry}	Time between each retry of SELECT command at NFC OP phase 3, until expiration of TO _{veh-4}	500ms

- 4 Note 1: The device might not respond to polling until the Key Tracking Receipt is received.

5 **6.3.7 Pairing Password URL**

6 To start owner pairing mode, the device may receive the pairing password through a URL. The
7 Vehicle OEM may provide the owner with the pairing password as fragment of a URL (e.g. via
8 an email). When the owner consumes this URL on the device, the device will redirect this link to
9 the Framework, which extracts the pairing password from the URL and uses it to start the owner
10 pairing. The URL is not intended to be reached out to. It is used to transport the parameters to be
11 interpreted by the framework.

12 The URL formatting rules are specified in RFC 3986 [\[27\]](#). The URL syntax shall take the
13 following form:

14 `scheme://authority/version/redirect?query#fragment`

15 For example:

<https://digitalkeypairing.org/v1/0001?technology=NFC,BLE&graphics=0001A0A1A2A3&authmethod=0001B0B1#pwd=12345678>

Table 6-6: Pairing Password URL Syntax

Component	Length (character)	Description	Field is
scheme	5	“https”: HTTP protocol operating over TLS	mandatory
authority	21	Domain name defined and registered by CCC Domain: digitalkeypairing.org	mandatory
version	2	“v1”: Version number to identify the fields in the URL	mandatory
redirect	4	Vehicle Brand Identifier [35]. This is used for server redirection in owner pairing.	mandatory
query		Non-sensitive URL parameters	mandatory
technology	variable	Communication interfaces supported for pairing by the vehicle. This parameter may be “NFC” or “NFC,BLE” If this query is not present, the communication interface is expected to be “NFC”	optional
graphics	12	The NFC reader position graphics identifier of the vehicle which is provided by vehicle OEM. This identifier is given as [Vehicle Brand Identifier 4-byte vehicle defined graphics identifier], represented as string. The Vehicle Brand Identifier is defined in [35].	optional
authmethod	8	The authorization method graphics identifier of the vehicle which is provided by vehicle OEM. This identifier is given as [Vehicle OEM Value 42-byte vehicle OEM defined authorization graphicsmethod identifier], represented as string. The Vehicle OEM Value is defined in [35].	optional
fragment		Sensitive URL parameter Device strips the fragment from the URL when using the URL to contact the authority to preserve the confidentiality.	mandatory
pwd	8	pairing password	mandatory

7 STANDARD TRANSACTION

The standard transaction protocol is intended to provide the following properties:

- Mutual authentication
- Forward secrecy
- Tracking resilience
- Integrity and confidentiality

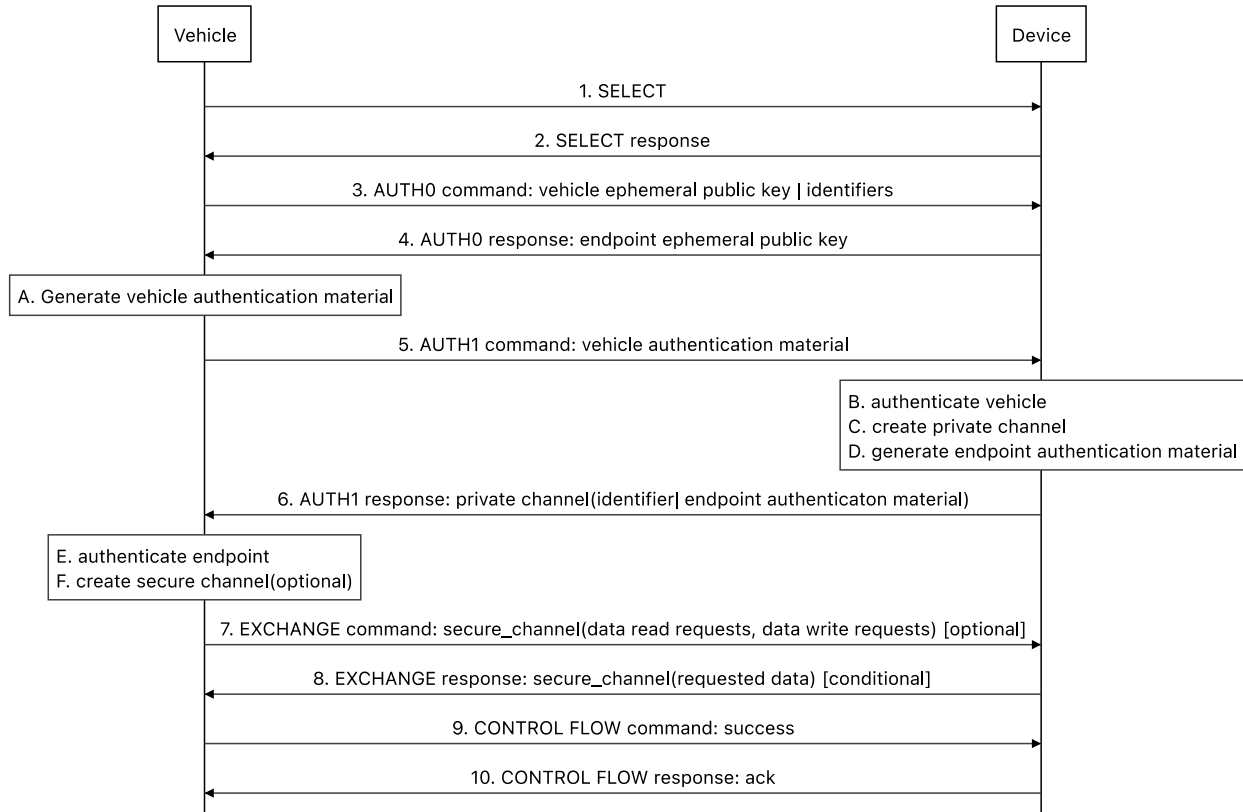
A secure channel between vehicle and device is initiated by generating ephemeral key pairs on the vehicle and device sides. Using a key agreement method, a shared secret can be derived on both sides and used for generation of a shared symmetric key, using Diffie-Hellman and a key derivation function.

The ephemeral public key generated on the vehicle side is signed with the vehicle's private key, vehicle_SK. This results in an authentication of the vehicle by the device. From the device's perspective, this guarantees that no privacy-sensitive data can be leaked by a MITM attack. This principle also allows the device to transmit data to the vehicle without any possibility of leakage by a passive or active eavesdropper.

Finally, the device uses the established secure channel to encrypt its public key identifier along with the signature computed on a vehicle's data-derived challenge and some additional application-specific data. This verification of the device's signature by the vehicle allows the vehicle to authenticate the device.

1

Figure 7-1: Standard Transaction Flow



2

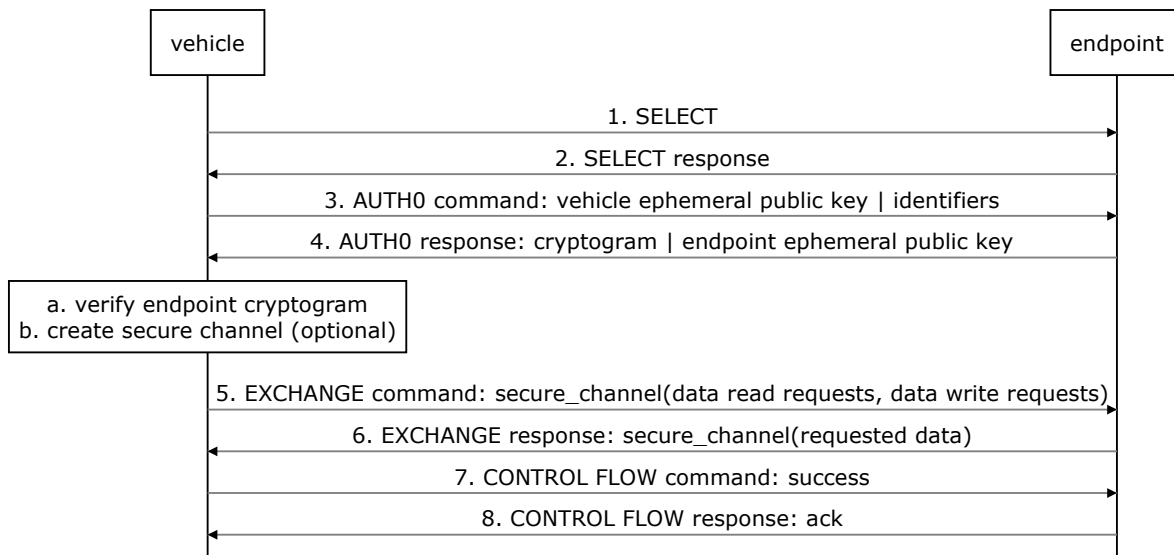
8 FAST TRANSACTION

The fast transaction protocol is intended to provide the following properties:

- Device authentication or Mutual authentication
- Integrity and confidentiality
- Tracking resilience

The device generates a cryptogram based on a secret previously shared during a standard transaction, and this allows the vehicle to authenticate the device. Optionally, a secure channel between vehicle and device is established by deriving session keys from a secret previously shared during a standard transaction and from the ephemeral keys. The ability of the vehicle to establish the secure channel authenticates the vehicle to the device. The shared secret shall always be regenerated in the standard transaction when fast transaction is allowed by the endpoint configuration defined during endpoint creation. Refer to Section [15.3.2.4](#) (see [Table 15-13](#))

Figure 8-1: Fast Transaction Flow



9 USER AUTHENTICATION

Devices shall provide the functionality to enforce user authentication according to their user authentication (UA) policies. The device shall provide at minimum one of the UA policies ([A], [AL], [E], [T]) (see [Table 9-2](#)). The user authentication policy is set by the user on the device. UA is enforced by the device only. The conditions for enforcing UA are determined by the selected UA policy, based on the transaction type (see [Table 9-2](#)).

The transaction type is set as a domain specific transaction_code value in P2 parameter of the AUTH0 command. Transaction type assignments are specified in [Table 9-1](#).

User authentication may be enabled or disabled individually for each Digital Key. The UA policy is stored as a list of transaction types that require user authentication. For every new Digital Key, the user authentication should be set to Off [O] by default (see [Table 9-2](#)). Device should inform Vehicle about the currently used UA policy at BLE connection setup, and also when UA policy is changed on the device (Communication of User Authentication Policy is described in Section 19.3.9.1).

Table 9-1: AUTH0 Command Transaction Type Coding

P2 (transaction_code)	Transaction Type
00 _h	RFU
01 _h	Door unlock
02 _h	Door lock
03 _h	First engine start authentication, first contact of a device with engine start rights in a vehicle session (e.g. door is in unlock state before the vehicle session starts and then first engine start in a new vehicle session)
04 _h	First engine start authentication, subsequent contact of a device with engine start rights in a vehicle session (e.g. door is in lock state before the vehicle session starts, and door is unlocked before first engine start)
05 _h	Other authentication request by vehicle
06 _h	User authentication request by vehicle
07 _h	First standard transaction at owner pairing (owner immobilizer token provisioning)
08 _h	Second standard transaction at owner pairing (read Key Tracking Receipt, provision receiver immobilizer tokens)
09 _h – 0F _h	RFU
10 _h	Derive ranging key (standard transaction only)
11 _h	First approach (standard transaction only), i.e. first standard transaction after Bluetooth LE pairing between vehicle and device outside of owner pairing
12 _h – FF _h	RFU

A vehicle session starts with the first usage of the Digital Key after the end of a preceding vehicle session and ends when the vehicle is locked or after an appropriate timeout of non-usage of the vehicle.

Table 9-2 shows the assignment of policies to the transaction types and the associated resulting user experience.

Table 9-2: User Authentication Policies

Device UA Policy														
Notation	Resulting User Experience	UA based on Transaction Code												
		00 _h	01 _h	02 _h	03 _h	04 _h	05 _h	06 _h	07 _h	08 _h	09 _h –0F _h	10 _h	11 _h	12 _h –FF _h
Off [O]	UA never requested	-	-	-	-	-	-	X	-	-	-	-	-	-
Access [A]	UA requested on first access or first engine start	X	X	-	X	-	-	X	-	-	X	-	-	X
Access [AL]	UA requested on first access and lock or first engine start and lock	X	X	X	X	-	-	X	-	-	X	-	-	X
Engine [E]	UA on every engine start	X	-	-	X	X	-	X	-	-	X	-	-	X
Always [T]	UA on every usage	X	X	X	X	X	-	X	-	-	X	-	-	X

- X: UA to be performed on the device
- -: UA not required for the transaction to be successful

The owner should not be able to restrict the UA setting options of a shared key.

Note: When user authentication is set for a Digital Key, the key may not be usable in Battery Low Mode.

9.1 Explicit User Authentication Policy

To improve security, it may be beneficial to require explicit UA before an action can be triggered. Explicit UA shall require an intent for RKE actions to be registered before explicit UA is enforced. The associated RKE action shall only be performed if the explicit UA has been performed successfully.

If the device leaves the state where only Digital Key functions can be triggered for up to 1 minute, then explicit UA shall not be required again when returning to that state to perform another RKE action.

Upon leaving this state where only Digital Key functions can be triggered, an explicit UA shall be performed before another action can be performed. Device should inform Vehicle if UA is not required because this state has been entered or if UA is required because this state has been left (Communication of User Authentication Policy is described in Section 19.3.9.1).

9.2 Implicit User Authentication Policy

To improve the user experience, it may be beneficial for the device to allow a single successful UA event to authorize several Digital Key actions within a short period of time (grace period). This may be coupled to a UA used for device unlock. If the user device implements such a grace period, it should not exceed 5 minutes. If supported by the device, additional successful UA events should cause this grace period timer to start/restart. The grace period shall expire immediately when the UA becomes invalid, such as when the device is locked. Device should inform Vehicle if UA is not required because the grace period has been entered or if UA is required because the grace period has been left (Communication of User Authentication Policy is described in Section 19.3.9.1).

This implicit UA shall not be used for critical actions, such as endpoint authorization, key sharing, or when a device's UA policy explicitly forbids implicit UA to be used for a transaction type or remote feature execution.

The user should be able to enable or disable usage of this implicit UA. The user may also be able to configure the length of the grace period between zero minutes (disabled) and the maximum time defined above.

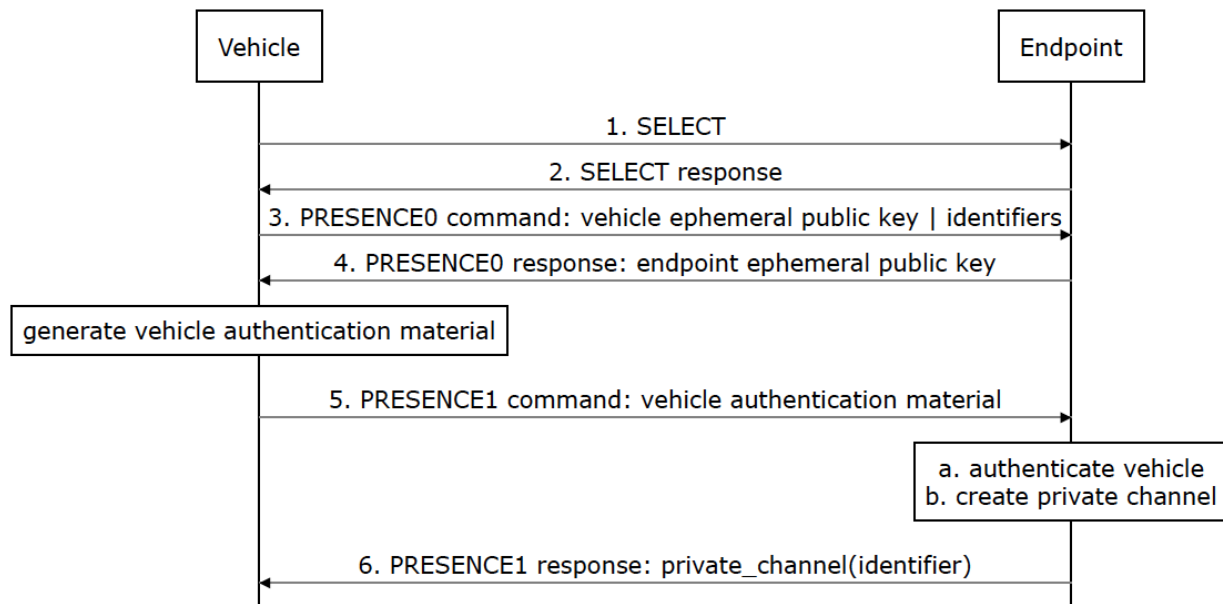
10 CHECK PRESENCE TRANSACTION

The check presence transaction protocol is intended to provide the following properties:

- Vehicle authentication
- Device identification
- Integrity and confidentiality
- Tracking resilience

The mechanism is similar to the standard transaction mechanism described in Section 7, except that the device signature is not sent to the vehicle, and user authentication is disabled. The goal is to allow verification of device presence near the vehicle without requiring user authentication, while preventing tracking.

Figure 10-1: Check Presence Transaction



11 DIGITAL KEY SHARING [WCC1/WCC2]

11.1 Encoding

All messages exchanged over the communication channels described in this section are compliant with Basic Encoding Rules-TLV(BER-TLV) as outlined in [40]. Certificates are compliant with X.509 format [3].

The TLV fields shall be ordered as described in this specification. A different field order is considered invalid unless specified otherwise. The nesting level is represented by indentation of tag values in the tag column.

11.2 Sharing Principles

11.2.1 Definitions

11.2.1.1 Sharing URL

To initiate the key sharing process, the sender device requests the Relay Server (see section 2.4.9) to create a Mailbox on the relay server. The relay server creates the mailbox, generates, and provides a sharing URL sender device. The sender device provides the sharing URL to the receiver device to initiate the transfer of the Digital Key. The sharing URL shall accept additional parameters. The version is determined by the relay server when generating the URL. The version is determined for use by the relay server and shall not have any impact on the sender or receiver device.

11.2.1.2 Activation Options

Can be defined by the Vehicle OEM to activate a shared digital key in the vehicle. The receiver can then choose any supported activation option that the Vehicle OEM decides to implement. Available Activation Options are listed in Table 17-51. At least one activation option with an additional fallback activation option should be enabled by the Vehicle OEM. If no activation option is advertised by the Vehicle OEM, device behavior is described in Section 11.2.2. This specification introduces “online sharing PIN” as a new activation option.

11.2.1.3 Approved Sharing Methods

These are Device OEM-proprietary sharing channels under full control of the device OEM that have been approved by the vehicle OEM. When approved as secure by a vehicle OEM then the sharing mechanism doesn't require activation options to be used for a key shared using that method. Vehicle OEMs shall have a way to define the activation method policy for approved sharing methods.

1

Table 11-1: Approved Sharing Methods

Tag	Length (bytes)	Description	Field is
7F10 _h	variable	list of approved sharing methods	
61 _h	variable	Tuple of first sharing method provider identifier and first sharing method group identifier	conditional
40 _h	2	Sharing method provider identifier as defined in XXX (e.g., 0x0000 = CCC, 0x0001 Apple, 0x0002 = Google, 0x0003 = OEM 3...)	mandatory
41 _h	variable	Sharing method group identifier defined by the sharing method provider	mandatory
61 _h	variable	Tuple of second sharing method provider identifier and second sharing method group identifier	optional
40 _h	2	Sharing method provider identifier as defined in XXX (e.g., 0x0000 = CCC, 0x0001 Apple, 0x0002 = Google, 0x0003 = BMW...)	optional
41 _h	variable	Sharing method group identifier defined by the sharing method provider	optional

2

3 11.2.1.4 Sharing Password

4 Sharing password is an activation option that is implemented as defined in [\[41\]](#).

5 11.2.1.5 Key Fob Activation

6 To verify that the presenter of a digital key is indeed the intended recipient of that digital key, a
7 Vehicle OEM may require the presence of a key fob during the First New Key Transaction.

8 11.2.1.6 Online Sharing PIN (OSP)

9 For this activation option the PIN is generated by the sender device and provided to the vehicle
10 OEM through a new server API preShare(). The sender provides the PIN to the receiver via a
11 dedicated channel, separate from the channel used to communicate the sharing invitation. The
12 receiver device sends the entered PIN to the vehicle OEM through another server API preTrack()
13 for verification. The sharing flow continues only if the PIN verification was successful.

14 11.2.1.7 Device PIN (DP)

15 The device PIN can be used by the sender to enhance sharing security if the vehicle OEM does
16 not mandate the use of an activation option. The sender device generates the device PIN and
17 sends it in the encrypted data through the relay server to the receiver device. The sender also
18 provides the PIN to the receiver via a dedicated channel, separate from the channel used to
19 communicate the sharing invitation. The receiver device then compares the PIN received through
20 the relay server to the value entered by the user on the receiver device. The vehicle OEM does
21 not verify the Device PIN. Implementation of this activation option is mandatory for device
22 OEMs.

11.2.1.8 Sharing Method Group Identifier

Identifier defined by OEM that implements a proprietary sharing method. Vehicle OEMs use this identifier to define the activation options policy for the referenced sharing method.

11.2.2 Primary Sharing Principles

The Digital Key system is based on signature verification using asymmetric cryptography. The vehicle unlocks the doors or starts the engine only if a challenge has been signed with a private key corresponding to a public key registered in the vehicle.

After the owner device and the vehicle have been paired as described in Section 6, the owner possesses a private key for which the corresponding public key is stored in the vehicle.

To allow keys shared by a device or a Server-Based device (SBOD, SBFD) to access the vehicle, the sender uses their private key to sign the receiver public keys. On presentation of the sender's signature over the receiver's public key, the vehicle, after successful verification of the sender's signature, will accept the receiver as a new vehicle user and store the receiver's public key.

Digital key sharing consists of the sender sending an invitation URL to the receiver device and a stateful exchange of key creation, key signing, and data import requests. When a secure sharing channel, such as a device OEM-controlled proprietary messaging channel, is used, no additional second factor verification (i.e., *activation options*) of the destination device should be required. The sender should have the option to use the device PIN for additional security. These secure sharing channels are named *approved sharing methods* (Section 11.2.1.3) throughout this specification, which means that they are approved by the vehicle OEM for use without additional activation options. The approval process is out of scope of this specification. The list of approved sharing methods (Table 11-1) can be pre-agreed between vehicle OEM and device OEM, or it can be sent using key tracking and event notifications.

Digital Key Sharing between devices of different device brands (known as cross-platform sharing), consists of:

1. Sending an invitation over a messaging channel that the sender uses regularly with the receiver. This provides good insurance that the invitation is received by the intended person. The channel to exchange the stateful sharing messages is defined by [38].
 - a. One or more *activation options*, (E.g., presence of a key fob during the First New Key Transaction with the receiver key), shall be applied if required by the vehicle OEM. The list of *activation options* for a given vehicle is communicated to sender and receiver device during key tracking. Vehicle OEM provides the list of *activation options* in the trackKey response or in case of a vehicle software update that adds or changes activation options, keyholder device can be updated using event notifications. Available *activation options* are listed in Section 17.11.1.17. In addition, the sender device provides the list of *activation options* to the receiver device during key sharing. The activation options shall be shown in the sender and receiver device. Receiver can select and use one option from the list of activation options.
2. If a vehicle OEM decides to not support additional *activation options*, the sender device should propose the use of a *device PIN* (and potentially other “silent” verification methods in the future) to senders that want to add additional verification to their shares. The policy for device OEM-proprietary sharing methods is defined by the sender device OEM.

In some cases, (e.g., when list of *approved sharing methods* has been modified by the vehicle OEM) the sender device shall send a *sharing method attestation* for shares using an approved sharing method to confirm that the sender device uses the updated list of *approved sharing methods* (see [11.2.1.3](#)).

The conditions of change of the sharing policy (i.e., adding or removing approved sharing methods from the list) by the vehicle OEM are out of scope of this specification.

The vehicle OEM shall notify the device OEM of a change to the activation policy using either an event notification “SHARING_POLICY_UPDATED” or another device OEM proprietary method.

11.2.3 *Sharing in a Chain Principles*

Sharing in a Chain (SiaC) feature described in this Digital Key specification covers all the use cases of daily usage of private vehicles and is the basis of introduction of (delegate) services for private vehicles and fleet vehicles.

The limited sharing principle in section [11.2.2](#) is broadened by configuring the option_group_1 of the Digital Key in a manner to allow re-sharing. The Sharing options, entitlements, and accountRoles are provided in a new uiBundle structure to the sender device ([17.11.1.12](#)). The shareableKeysCount in uiBundle indicates the total number of remaining keys that can be shared while sharedKeysCount in uiBundle indicates the total number of keys that have already been shared. The number of times re-sharing can occur is determined by the account role assigned to a shared key by the sharer and is controlled by the Vehicle OEM server which provides the list of supported accountProfiles and accountRoles in the UiBundle.

SiaC requires slot identifiers and optionally immobilizer tokens to be provided online by the vehicle server. This is already a supported option in [\[41\]](#) and is mandatory in this specification. SiaC also requires a new method to obtain BLE key material online from the vehicle server because non-owner sender devices do not have the SPAKE2+-derived secrets to calculate BLE keys.

Existing shared keys can be migrated ([2.10.5](#)) to a new accountRole (see section [2.8.4](#)) to enable re-sharing. This occurs as soon as devices and vehicle software are updated to support SiaC. Owner and receiver endpoints are converted ([2.10.11](#)) on the applet level to allow re-sharing, which is not allowed in prior versions of Digital Key specification. Note that keys that do not contain the required authorized_PK in their endpoint data cannot be migrated.

Receivers of shared and re-shared keys can either be devices that support SiaC or legacy devices that support Digital Key specification version 1.0 or 1.1. These legacy devices can only receive valid BLE keys from an owner device. If the keys are shared from non-owner devices, then the legacy devices will not have valid BLE keys as a consequence these legacy devices can only use NFC Digital Keys.

To receive functional BLE keys from non-owner sender devices, it is necessary to update legacy devices to support the Digital Key Specification 1.1.4, which describes a method for legacy devices to receive BLE keys online from the vehicle OEM server.

All keys on the same device OEM account, which is identified by the groupIdIdentifier, shall have the same Access Profile ([Table 11-21](#)) and accountRole ([Table 2-3](#)) per vehicle.

A device shall only have one Digital Key for a vehicle. The vehicle OEM server shall reject key tracking requests with the appropriate error substatus code (see [Table 17-68](#)) for keys shared from a different sender to any device on an account where one device already has a key for the same vehicle.

11.2.4 Server-Based Key Sharing Considerations

After the owner device and the vehicle have been paired as described in Section 6, or the vehicle has been infleeted with an owner server (SBOD) as described in section 12.2, the owner entity possesses a private key for which the corresponding public key is stored in the vehicle. The public key of the KIS can be stored in the vehicle during production or provisioning. Alternatively, the SBOD PK can be used for the KIS

The signature of keys shared by an SBOD which is hosted by the vehicle OEM might be verified by the KTS and the fleet vehicle might accept a shared key on presentation of the KTS signature only.

When delegate services are allowed, vehicle OEMs must provide an option for the vehicle owner and eligible users and their devices to control sharing of keys for their vehicle, as described in Section 12.4.6 Owner Sharing Control.

11.3 Communication Channel

The standardized communication channel between devices and the relay server used by devices and vehicles that implement cross-platform sharing shall use the stateful workflow method defined in [\[38\]](#).

The referenced sharing channel has the following properties:

- Open standard for implementation on all Digital Key eligible devices and servers
- Sharing invitation can be sent over any messaging or chat channel.
- Devices connect directly to relay server.

In addition, the devices shall only send encrypted sharing data using the encryption key from the sharing invitation.

The communication channel between a sender device and a receiver device from the same OEM is implementation-specific and out of scope of this specification but shall be listed as an approved sharing method in [Table 11-1](#), if key activation without vehicle OEM-defined second factor activation option is intended. [Figure 11-1](#) shows the detailed key sharing flow between the sender device and receiver device. The process follows the “Stateful Workflow” defined in [\[38\]](#)

Note: In [Figure 11-1](#), the full head arrows represent synchronous requests, dashed arrows represent responses to synchronous requests and open head arrows represent asynchronous messages

[Figure 11-1](#) does not show all steps or parameters, such as the full *payload* coding or the device OEM server. It is meant to clarify the flow in principle and points to the relevant elements.

The exchanges between sender and receiver devices are described by the stateful workflow method using the following HTTP access methods as defined in [\[38\]](#). An application running on a device may invoke the following APIs on Relay Server (see Section [11.3.4.1](#) to [11.3.4.6](#))

- CreateMailbox
- UpdateMailbox
- DeleteMailbox
- ReadDisplayInformationFromMailbox
- ReadSecureContentFromMailbox
- Relinquish Mailbox

The secure element is not involved in generating key material used for the sharing channel as described in this section.

The key sharing flow and data is described in Section 11.4.

11.3.1 Notifications

The push notification feature shall be implemented by the Relay Server.

Notification tokens as described in [\[38\]](#) should be used by every device OEM.

If the notification tokens are not used, the devices shall implement a polling strategy for Relay Server data. Devices may implement a (backup) polling strategy even if push notifications are used.

The polling strategy should follow these recommendations:

- After sending the invitation, the owner device should poll every 10 seconds for a duration of 3 minutes for the Signing Request (or other messages to be received as per [Table 11-4](#)), thereafter it should poll every 30 seconds for the next 10 minutes, thereafter it should poll every 3 minutes for the next hour, thereafter every hour until the mailbox is expired or deleted.
- After accepting the invitation and uploading the signing request to the mailbox, the receiver device should poll every 5 seconds for a duration of 1 minute for the Import Request (or other messages to be received as per [Table 11-4](#)), thereafter it should poll every 30 seconds for the next 10 minutes, thereafter it should poll every 3 minutes for the next hour, thereafter every hour until the mailbox is expired or deleted.

11.3.2 Cross-Platform Sharing Invitation

The sender device generates a Secret and encrypts the payload for the CreateMailbox API using this Secret as described in [\[38\]](#).

The sender device then initiates the sharing session by calling the CreateMailbox API as described in Section [11.3.4.1](#). The call returns the sharing invitation in form of a URL. The URL

is then sent to the receiver over any communication channel (e.g., WhatsApp, etc.). The URL formatting is described in [38]. The framework shall append the Secret as a fragment to the URL.

A relay server may be implemented by any CCC member. All CCC-approved relay server URLs for device OEMs and SBxD/KIS providers shall be listed in [35]. The approval process is defined in a separate document and is controlled by the CCC. All sender devices and servers shall only use approved relay server URLs. Receiver devices and servers shall only accept invitations containing an approved relay server URL. Contingent on the establishment of business relationships between device OEMs, SBxD/KIS server providers and relay server providers, device OEMs and SBxD/KIS server providers should support all CCC-approved relay server URLs.

If the relay server is used for sharing keys to a device (e.g., fleet management), then receiver devices should support all CCC-approved relay server URLs within a reasonable time frame as defined by the CCC in the CCC Program Management Document [PMD].

A receiver device may be unable to accept the sharing invitation for many reasons, such as incompatible device etc. In such cases, the receiver device should be redirected to a landing webpage in the sender domain that offers at least generic, ideally device OEM-specific, guidance to the user of receiver device. This guidance can include, but not limited to, enumerating the possible failure reason, providing a list of supported devices, and device OS versions, providing corrective next steps, etc. The landing webpage is maintained by the sender device OEM.

11.3.3 General API Parameter Definitions

The *version* parameter shall be set as described in [38].

The *mailboxIdentifier* shall be used as described in [38].

The *deviceAttestation* should be used to authenticate the sender device if the Relay Server is not hosted by the owner device OEM. In other cases, the device OEM might use proprietary methods to authenticate the device. The receiver device does not need to present a *deviceAttestation*.

The *deviceClaim* shall be provided and set by sender and receiver device as described in [38].

The payload is encrypted as described in [38]. For Digital Keys payload shall contain the provisioning information as the JSON structure with the keys *format* and *content* as per [38]. For Digital Keys the format identifier is *digitalwallet.carkey.ccc*. The *content* data structure is described in Table 11-2.

Table 11-2: content in JSON Format

Key	Type	Max Length (Bytes)	Description	Required
genericSharingData	Dictionary	4096	This is data shared between devices on different device OEM platforms.	Yes
<Device OEM>	Dictionary	4096	Device OEM proprietary provisioning information for the designated device OEM. Ignored by all other device OEMs.	Optional

<Device OEM> is defined in [35].

1 The sender device can add proprietary data (e.g., “apple”) that a receiver device from the same
2 device OEM is able to use, e.g., to improve the user experience. If the receiver device is from
3 another device OEM, then the receiver device shall ignore the proprietary data. It shall be
4 possible for the owner device to add multiple <Device OEM> fields, each one for a different
5 device OEM, if required.

6 Note that all fields can be “seen” by receiver devices of all platforms.

7 The genericSharingData is defined as the following JSON structure:

8 *Table 11-3: JSON formatted genericSharingData data structure*

Key	Type	Max Length (bytes)	Description	Required
sharingData	String	4096	Sharing data structure (TLV string) as listed in Table 11-4 based on the value of sharingDataType	Yes
sharingDataType	Integer	4	Enum for type of message. See sharingDataType in Table 11-4 .	Yes
sharingId	String	64	ID of sharing invitation	Yes
friendKeyId	String	64	Identifier of key on receiver device (this is used on owner side to track shared keys)	Conditional. Required for sharingDataType sharingKeySigningRequest.
authType	Array of strings	variable	“VehicleActivation” = vehicle OEM-enforced activation options. See Table 17-62 “DevicePIN” = PIN entered on receiver device as per [41] Other receiver verification options can be added in the future.	Conditional. Required for sharingDataType sharingKeyCreationRequest and if an additional verification method is required by sender Device or Vehicle OEM
activationOptions	Array of strings	variable	List of activation options as per section 17.11.1.17	Conditional. Required for sharingDataType sharingKeyCreationRequest if authType = “VehicleActivation”.
sharingPasswordLength	Integer	4	Length of PIN/password applicable for activationOptions: OSP and sharingPassword. Shall be ≥ 4 and ≤ 12 .	Conditional. Required for sharingDataType = sharingKeyCreationRequest and if authType=“VehicleActivation”.
pinLength	Integer	4	Length of device PIN. Absent if device PIN is not used.	Conditional. Required for sharingDataType sharingKeyCreationRequest and if authType = “DevicePIN”.

Key	Type	Max Length (bytes)	Description	Required
brand	String	variable	Brand of the vehicle	Conditional. Required for sharingDataType sharingKeyCreationRequest
model	String	variable	Model of the vehicle	Conditional. Required for sharingDataType sharingKeyCreationRequest
vehicleIdentifier	String	variable	Value of the vehicle_identifier	Conditional. Required for sharingDataType sharingKeyCreationRequest
...unknown additional entries shall be ignored, reserved for forward compatibility.				

sharingData contains the key sharing data elements as defined in [Table 11-4](#), depending on the state of sharing.

sharingDataType is defined in [Table 11-4](#).

sharingId is defined by the sender device in the first message to unambiguously identify the sharing session independently of the mailboxIdentifier. The value shall be unique per sender device and per session. *sharingId* is generated by calculating a SHA-256 hash over the key creation request ([Table 11-5](#)). sender and receiver device shall use the same value in all messages of the same sharing session. If the sender sends out multiple invitations (e.g., for receiver phone and receiver watch) then the values of *SharingId* shall be unique per receiver device.

Note: Testvector for Hash of Key Creation Request needs to be added

friendKeyId shall be sent by the receiver device in the key signing request (format as defined in section 2.4 as “digital key identifier”)

authType indicates which of the following additional verification is required. If no method is used, then the field shall be absent. The additional verification methods are listed here:

- *VehicleActivation* usage shall be indicated if both of the following conditions are true:
 - At least one activation option has been provided to the sender device via uiBundle in trackKey response (see section [17.7.3.3](#)) or eventNotification (see section [17.9.1](#))
 - A non-approved sharing method has been used
- *DevicePIN* usage shall only be indicated if both of the following conditions are true:
 - No vehicle activation is required
 - The sender chooses to use the device PIN for a key sharing

It shall be possible to add other receiver verification options, which are not part of the current list of activationOptions, in the future.

activationOptions are provided by the vehicle OEM in the trackKey response ([17.7.3.3](#)) or eventNotification. Those options are specific to and implemented by the vehicle and may include Online Sharing PIN. Sender and receiver device shall indicate them in the UI. If the field is

absent, then no additional verification is required by the vehicle OEM. In this case, where no activationOptions are provided by Vehicle OEM, sender or device OEM can still require other verification methods not involving the vehicle OEM, as described in authType.

sharingPasswordLength shall be a value of length ≥ 4 and ≤ 12 , chosen by the sender device. A suitable combination of PIN/password length and allowed number of attempts to enter the device PIN/password in the receiver device UI shall be chosen by the owner device, following the recommendations in [38].

pinLength shall be a value between 4 and 8 (including 4 and 8), chosen by the owner device. A suitable combination of PIN length and allowed number of attempts to enter the device PIN in the friend device UI shall be chosen by the owner device, following the recommendations in [38].

brand is the string describing the vehicle brand as received by the sender device in trackKey() Response (as per section 17.7.3.3).

model is the string describing the vehicle model as received by the sender device in trackKey() Response (as per section 17.7.3.3).

vehicle_identifier is the value as provided in the endpoint configuration (as per section 15.3.2.4 tag 7F27_h, subtag 4D_h).

Table 11-4: *sharingDataType* enum Definition

Key	Value
sharingKeyCreationRequest (Key Creation Request, see Table 11-5)	1
sharingKeySigningRequest (Key Signing Request, see Table 11-6)	2
sharingImportRequest (Import Request, See Table 11-11)	3
sharingSenderCancel	4
sharingReceiverCancel	5
sharingPinReEntryRequest (Table 11-16)	6
sharingPinReEntryValue (Key Signing Request, see Table 11-6)	7
...unknown additional entries shall be ignored, reserved for forward compatibility.	

The sharing types *sharingSenderCancel* and *sharingReceiverCancel* shall be used on any error condition instead of the next message of the sharing flow and shall contain an error message (see Table 11-26) with the appropriate error code.

The sharing type *sharingPinReEntryRequest* signals the receiver that the entered device PIN was wrong. It contains the number of remaining attempts in the field *sharingData* in TLV format.

The sharing type *sharingPinReEntryValue* contains the key signing request with the new receiver device PIN.

Sample Provisioning Information in CreateMailbox Message is shown in Listing 11-1.

Listing 11-1: Sample Provisioning Information in JSON Format

```
{
  "format":"digitalwallet.carkey.ccc",
```

```
3  "content": {  
4    "genericSharingData": {  
5      "sharingData": "7F31xxx",  
6      // key creation request, this field changes depending on the sharingDataType value  
7      "sharingDataType": 1,  
8      "sharingId": "123456789..",  
9      "friendKeyId": "A2B686DFC..  
10     "authType": ["DevicePIN"]  
11     "pinLength": 6  
12   },  
13   "apple": {  
14     // ...  
15   }  
16 }  
17 }
```

11.3.4 API Parameter Usage

The usage of API parameters from [38] that are relevant to Digital Key Cross Platform Sharing are described in the following sections.

11.3.4.1 CreateMailbox API Parameters

The payload data contains *format* and *content* fields (see [38]). *content* contains *genericSharingData* and optional device OEM-specific sharing data as described in Table 11-2. *sharingDataType* enum (Table 11-4) shall be set to 1 (sharingKeyCreationRequest) to indicate that the content of *sharingData* is the key creation request as defined in Table 11-5.

The *displayInformation* is determined by the sender device. The format is described in [38].

The *notificationToken* is provided by sender device at the time of mailbox creation and is valid for the lifetime of the mailbox. It should be used by the sender device to receive a notification from the relay server when a Key Signing Request, sharing rejection or subsequent retry of device PIN (in context of Key Signing Request) is received by the Relay Server from the receiver device. If notifications are not used, then the sender device shall perform regular polling as described in Section 11.3.1.

The *mailboxConfiguration* shall be set as follows:

- *accessRights* = “RWD”
- *expiration* = <owner device OEM policy>

The *urlLink* is formatted as described in [37].

isPushNotificationSupported shall be set to “true” by all CCC relay servers, i.e., they shall be supported by servers. Devices may choose to not use them although their use is strongly recommended.

11.3.4.2 UpdateMailbox API Parameters

The payload contains *format* and *content*. *content* contains *genericSharingData* and device OEM-specific sharing data as per Table 11-3.

sharingData from a receiver device contains

- the Key Signing Request as per Table 11-6, or
- a *sharingReceiverCancel* error message.

1 *sharingData* from a sender device contains one of the following:

- 2 • the Import Request as per [Table 11-11](#), or
- 3 • a *sharingSenderCancel* error message, or
- 4 • the Device PIN entry request message

5 The *notificationToken* should be provided by the receiver device to be notified when data has
6 been received by the Relay Server from the owner. The parameters are defined by the receiver
7 device. If notifications are not used, then the receiver device shall perform regular polling as per
8 Section [11.3.1](#).

9 *11.3.4.3 DeleteMailbox API Parameters*

10 The receiver device shall delete the mailbox after successfully retrieving the Import Request.

11 If the sender receives a sharing cancellation from the receiver (e.g., receiver rejects the
12 invitation), the sender device shall delete the mailbox.

13 If the receiver receives a sharing cancellation from the sender (e.g., sender does not sign the
14 receiver key), the receiver device shall delete the mailbox.

15 If the sender wants to cancel a sharing after the key has been signed, in addition to deleting the
16 mailbox a Remote Termination Request shall be used as outlined in [Figure 13-4](#) [REV_140]. The
17 vehicle OEM shall add the receiver key ID to a deny-list and reject key tracking (using substatus
18 code 50112), if the receiver key is not yet registered.

19 *11.3.4.4 ReadDisplayInformationFromMailbox API Parameters*

20 The *displayInformation* posted during mailbox creation can be retrieved again if required.

21 *11.3.4.5 ReadSecureContentFromMailbox API Parameters*

22 The payload contains the payload uploaded during creation or update of the mailbox according to
23 the descriptions in the respective sections.

24 The *displayInformation* posted during mailbox creation is retrieved and used to display a
25 placeholder for the final key graphics on the receiver device.

26 The *expiration* of the mailbox indicates the maximum lifetime of the mailbox if it is not deleted
27 before.

28 *11.3.4.6 RelinquishMailbox API Parameters*

29 This API might be called to allow a device (e.g., a tablet) that receives the sharing invitation but
30 cannot handle a digital key, to release the mailbox link and let another device connect to the
31 mailbox as outlined in [\[38\]](#). This allows another recipient to claim the mailbox and use the
32 appropriate device to proceed with sharing procedures.

33 *11.3.5 Security Considerations*

34 The security considerations in [\[38\]](#) are generic. Some considerations how to apply them in the
35 context of Digital Key:

- 36 - The slot identifier ensures that a key creation request can lead to the creation and
37 acceptance of only one key based on this request.

- If not using an approved sharing method, sharing should be protected by a second factor, which can be either a vehicle OEM-defined second factor (i.e., activation option) described in section [11.2.1.2](#) and later, if supported by the vehicle, or a device OEM-defined second factor (e.g., the device PIN) as described in section [11.2.1.6](#).
- Sharing shall require explicit user authentication (as described in section [9.1](#) for RKE). Explicit user authentication is the combination of user intent and user authentication.

11.3.5.1 Device Requirements

Sender device shall meet the following requirements:

1. A cryptographically secure Deterministic Random Bit Generator (DRBG) (see NIST Special Publication 800-90A [\[42\]](#)) with a reliable source of entropy (see NIST Special Publication 800-90B [\[43\]](#)).

Receiver device shall meet the following requirements:

1. When using Online Sharing PIN (OSP), after reaching `attemptsLeft = 0`, the created endpoint and all associated data shall be deleted from receiver device

11.3.5.2 Vehicle OEM Server Requirements

Vehicle OEM backend infrastructure shall meet the following requirements

1. Minimum number of digits used for Online Sharing PIN shall be 6 (as per NIST Special Publication 800-63B [\[39\]](#)). For ease of use, maximum length of Online Sharing PIN shall be 8.
2. To prevent a successful brute force attack within feasible time, the number of retries shall be less than 10 (as per NIST Special Publication 800-63B [\[39\]](#))
3. For a given sharing attempt (uniquely identified by the `sharingId` and bound to the sender by `sharingId` and endpoint PK), any server API calls (including `preShare`, `preTrack`, `trackKey`) after a successful attempt, but with different content in the call shall be declined. The sharing attempt shall be disallowed and data pertinent to the sharing attempt that does not impact existing keys shall be deleted.
4. For each `sharingId`, once the server backend receives a `preTrack` request signed by a certain receiver PK, `preTrack` requests from other receiver PKs for the same `sharingId` shall be declined. If this occurs the sharing session shall be cancelled and data pertinent to the sharing attempt that does not impact existing keys shall be deleted.
5. Sharing sessions (uniquely identified by the `sharingId`) for the same vehicle initiated by the same sender device (bound by `sharingId` and endpoint PK) shall be kept independent, even if the sessions are executed in parallel. For example, multiple `preTrack` calls before final `trackKey` by different receiver devices for the same vehicle for the same sender.
6. When PIN verification for a given receiver endpoint PK fails (`attemptsLeft = 0`) then this endpoint and the associated `sharingId` shall not be accepted anymore until sender endpoint is deleted (terminated).
7. The key tracking server shall verify that the account Role of the key being tracked is in the allowed list of account roles originally given to the sharer key.

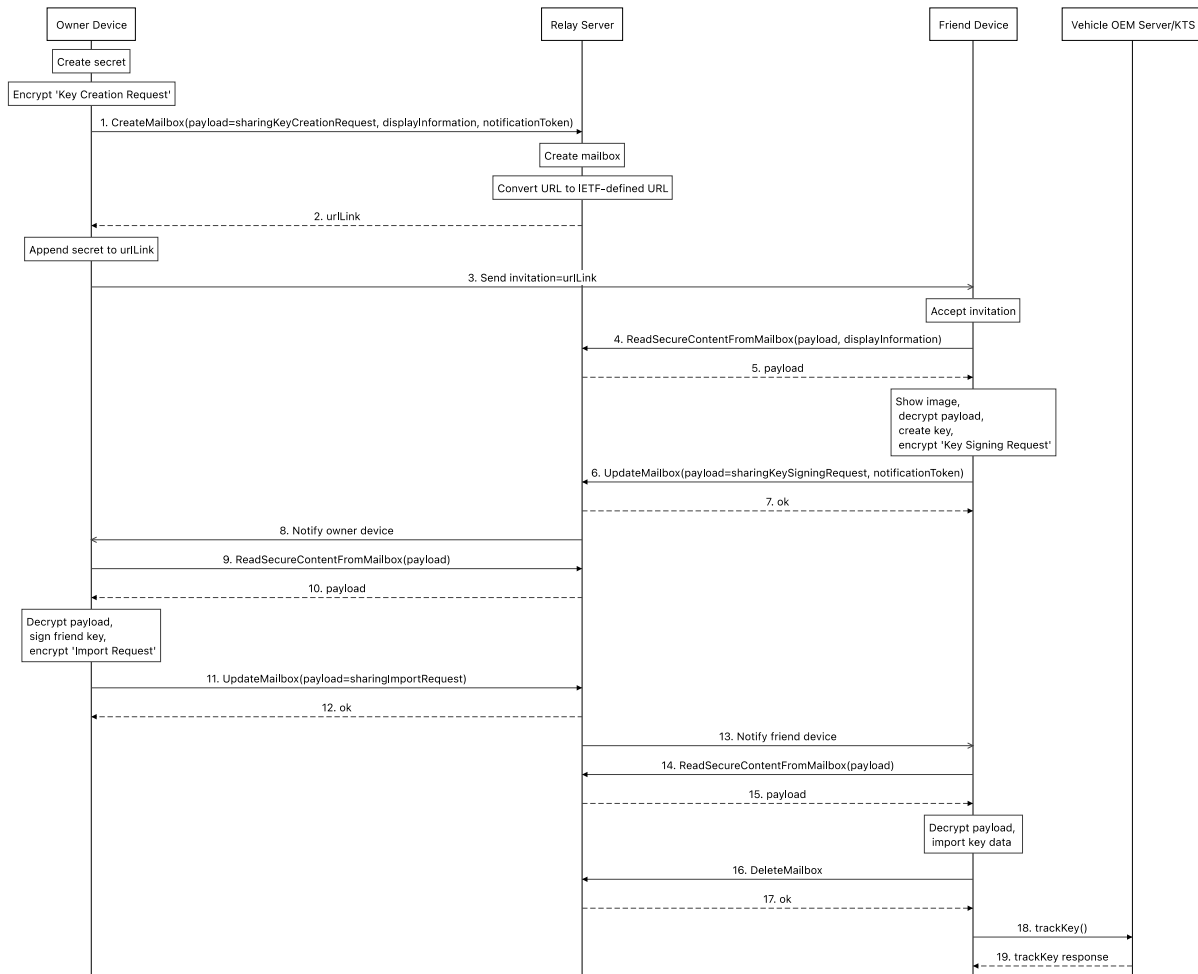
11.4 Inter-Account Sharing Key Sharing Flow: Steps

[Figure 11-1](#) shows the main key sharing flow and describes the content of messages exchanged between the sender and receiver devices independently of the communication channel or transport layer used to convey these packets.

At the end of this flow, the receiver's private mailbox contains an attestation package in which the receiver's public key is signed by the sender private key along with a set of entitlements. The receiver's confidential mailbox optionally contains an active immobilizer token.

The last step describes how the attestation package is delivered and verified by the vehicle during the First New Key Transaction.

Figure 11-1: Detail of Key Sharing Flow Between Sender and Receiver device After Channel is Established



11.4.1 Steps 1 through 5 (Sender): Sharing Invitation

The sender device sends a Key Creation Request ([Table 11-5](#)) containing the required endpoint configuration and the entitlements to be granted. The endpoint_identifier, instance_CA_identifier

- 1 and account_info_hash from field endpoint_configuration shall not be included since their
- 2 content is defined by the receiver device.

3 *Table 11-5: Key Creation Request*

Tag				Length (bytes)	Description	Field is	Domain Version
7F31 _h				variable	Key Creation Request		N/A
	endpoint_configuration data field as described in Table 15-13 (without fields endpoint_identifier, instance_CA_identifier and account_info_hash), with Tag 7F2C _h instead of Tag 7F27 _h for receiver devices that support OD-FD-KS = 0300 _h					mandatory	V-OD-FW
	endpoint_configuration data field as described in Table 15-13 (without fields endpoint_identifier, instance_CA_identifier and account_info_hash), with Tag 7F27 _h as per [41] for receiver devices that support OD-FD-KS < 0300 _h or no explicit OD-FD-KS versioning					mandatory	V-OD-FW
	key configuration data field in ASN.1 as described in [41]					mandatory	V-OD-FW
	key_configuration data field (TLV) as described in Table 11-21					mandatory	V-OD-FW
	92 _h			8	Random to ensure sharingId calculated by generating SHA-256 hash of Key Creation Request is unique	mandatory	N/A
	59 _h			variable	ROUTING_INFORMATION as generated during owner pairing described in Section 5.1.4.3	mandatory	V-OD-FW
	5B _h			2 x n	V-OD-FW-vehicleList (ver.high ver.low), agreed version first as obtained during owner pairing described in Section 6 (This is the vehicle owner device framework version list in SPAKE2+ REQUEST during owner pairing).	mandatory	V-OD-FW
	5C _h			variable	Digital Key applet transaction (V-D-TX) protocol versions as obtained during owner pairing described in Section 6 (This is the Digital Key applet protocol versions in which the owner received in SPAKE2+ REQUEST during owner pairing)	mandatory	V-D-TX
	7F49 _h			variable	Vehicle wireless capabilities. Subtags D0 _h , D1 _h , D3 _h and D4 _h shall not be included when this tag is transmitted within a Key Creation Request. See Table 19-90	mandatory	V-OD-FW
	7F60 _h			variable	Sharing configuration derived from sender SHARING_CONFIGURATION as generated during owner pairing described in Table 5-14 or received from vehicle server	mandatory	N/A

Tag				Length (bytes)	Description	Field is	Domain Version
		DA _h		1	If this tag is present, receiver device shall obtain immobilizer token and/or slot identifier. The following values are assigned: 01 _h immobilizer token and slot identifier retrieved online 03 _h no immobilizer token required; slot identifier retrieved online Values 00 _h and 02 _h are deprecated.	mandatory	V-OD-FW
		DB _h		0	If this tag is present, receiver device shall obtain a key tracking receipt as per Sections 6.3.5 and 11.4.5	mandatory	V-OD-FW
		DC _h		0	If this tag is present, Vehicle OEM supports online attestation delivery feature as described in Section 11.10	optional	V-OD-FW
		DD _h		1	MAX_KEY_ATT_COUNT, number of attestation packages that can be verified in one First New Key Transaction. The minimum value is 2. This value shall be equal to or smaller than the number of additional KeyAtt sections in private mailbox (see section 4.3.1).	mandatory	V-OD-FW
		DE _h		0	If this tag is present, the vehicle supports key update attestations. In this Digital Key specification version, this tag shall be absent, but devices shall not fail if it is present in future Digital Key specification versions.	optional	V-OD-FW
...unknown tags under 7F60 _h shall be ignored, reserved for forward compatibility							
	4A _h			3	2 bytes offset (unsigned big-endian) 1 byte length to be returned from confidential mailbox in AUTH1 response as obtained during owner pairing in Section 6	optional	V-OD-FW
	4B _h			3	2 bytes offset (unsigned big-endian) 1 byte length to be returned from private mailbox in AUTH1 response as obtained during owner pairing in Section 6	optional	V-OD-FW
	D7 _h			1	2nd Factor activation required (00 _h =not required, 01 _h =required for non-approved sharing methods, other=RFU)	optional	V-OD-FW

Tag				Length (bytes)	Description	Field is	Domain Version
	D8 _h			1	SHARING_PASSWORD_LENGTH (vehicle OEM specific as per appendix B.1 . If absent, sharing password (Section 11.2.1.4) is not required. Also defines length of Online Sharing PIN (same as sharing password).	optional	V-OD-FW
	54 _h			2 x m	m supported OD-FD-KS compatibility versions of owner device (ver.high ver.low), ordered highest to lowest	mandatory	OD-FD-KS
	5E _h			Var.	V-D-BT DK Protocol Version lists of vehicle (as per Table 5-4)	conditional	V-D-BT
	80 _h			2 x n	Supported_DK_Protocol_Version (See Section 2.10.3.3)	conditional	V-D-BT
	9F17 _h			1	Number of PIN entry attempts. Shall be greater than 0 and less than 10	Conditional. Required if device PIN or Online Sharing PIN is used.	OD-FD-KS
	9F18 _h			1	Device PIN Length (in digits)	Conditional. Required if devicePIN is used	OD-FD-KS
	9F20 _h			9	Content of HU_PP message (Table 19-65)		V-D-BT
	60 _h			variable	Endpoint Configuration data for V-OD-FW = 0300 _h . Following data elements are different from V-OD-FW = 0100 _h		
		D5 _h		1	MBX_VERSION = 80 _h	mandatory	
		4A _h		3	2 bytes offset (unsigned big-endian) 1 byte length to be returned from confidential mailbox in AUTH1 response as obtained during owner pairing in section 6	Optional	
		4B _h		3	2 bytes offset (unsigned big-endian) 1 byte length to be returned from private mailbox in AUTH1 response as obtained during owner pairing in section 6	Optional	
	7F2B _h			variable	Concatenation of key server certificate chains and/or sender device certificate chains • SBxD/KIS certificate chain • Device certificate chain	mandatory	OD-FD-KS
		30 _h		variable	Certificate chain of first sender, either owner device or SBxD/KIS certificate chain	mandatory	OD-FD-KS

Tag				Length (bytes)	Description	Field is	Domain Version
			7F20 _h	variable	External CA certificate as per Listing 15-13 from the sender endpoint	conditional	V-OD-FW
			7F22 _h	variable	Instance CA certificate as per Listing 15-16 from the sender endpoint	conditional	V-OD-FW
			7F24 _h	variable	Endpoint Creation certificate as per Listing 15-5 from the sender endpoint	conditional	V-OD-FW
					Alternatively, if the first sender is an SBOD or KIS		
			7F20 _h	variable	External CA certificate as per Listing 15-13 from the sender endpoint	conditional	V-OD-FW
			7F42 _h	variable	SBOD/KIS intermediate certificate as per Listing 11-13 from the sender server	conditional	
			7F44 _h	variable	SBOD/KIS endpoint creation certificate as per Listing 11-10 from the sender endpoint.	conditional	
		31 _h or 7F44 _h	7F24 _h	variable	Second endpoint creation certificate [H] as per Listing 15-5 from the sender endpoint	Conditional. Present, if further sharing has occurred	
		32 _h or 7F44 _h	7F24 _h	variable	Third endpoint creation certificate [H] as per Listing 15-5 from the sender endpoint or SBOD/KIS endpoint creation certificate [T] as per Listing 11-13 from the sender server	Condition. Present, if further sharing has occurred	
		33 _h or 7F44 _h	7F24 _h	variable	Fourth endpoint creation certificate [H] as per Listing 15-5 from the sender endpoint or SBOD/KIS endpoint creation certificate [T] as per Listing 11-13 from the sender server	Condition. Present, if further sharing has occurred	
		3x _h		variable	x+1 th endpoint creation certificate [H] as per Listing 15-5 from the sender endpoint or SBOD/KIS endpoint creation certificate [T] as per Listing 11-13 from the sender server	Condition. Present, if further sharing has occurred	
...unknown additional tags under 7F31 _h shall be ignored, reserved for forward compatibility							

- 1
- 2 If tag D7_h is absent, then no activation option is required.
- 3 If tag D7_h is absent, then tag D8_h shall also be absent.
- 4 If the activation option enabled by the vehicle is sharing password, then tag D8_h is used to
- 5 provide the sharing password length to receiver devices. If the receiver is a SiaC-capable device

(D-VS = 0300_h) and if the activation option enabled is OSP, then tag D8_h refers to the online sharing PIN length. This means that sharing password and online sharing PIN shall be generated with the same length by the sender. **Note:** Information in Tag D7_h could be overwritten by the presence of the activationRequired field in the key tracking response and/or eventNotification.

The first sender (n=1) of a key is either the owner device or SBOD/KIS. Second sender and beyond are either non-owner device or SBFD. The Digital Key sharing chain starts from the first sender.

If the first sender is an owner device, then it shall provide the following in Tag 7F2B_h/30_h:

1. External CA certificate [F],
2. Instance CA certificate [E], and
3. Owner Digital Key certificate [H].

If the first sender is SBOD/KIS, then it shall provide the following in Tag 7F2B_h/30_h:

1. External CA certificate [F],
2. Key Server Intermediate certificate [S] (optional), and
3. SBOD/KIS Digital Key certificate [T].

If the nth sender (n > 1) is a non-owner device, then it shall provide in Tag 7F2B_h/3x_h (where x = n – 1), its device Digital Key certificate [H] only.

If the nth sender (n > 1) is a SBFD, then it shall provide in Tag 7F2B_h/3x_h (where x = n – 1), its SBFD Digital Key certificate [T] only.

The receiver of Key Creation Request from any sender shall verify

- that tag 7F2B_h exists in the Key Creation Request ([Table 11-5](#))
- that tag 7F2B_h/30_h is populated with a verifiable certificate chain and verify the chain
- that the vehicle class ([Table 15-16](#) - Tag 47_h, bit 3) indicated in the endpoint creation data matches the vehicle class indicated in the endpoint certificate sent within Tag 30_h
- that the signature on the attestation package received in the import request can be verified using the public key from the endpoint certificate on the highest 3x_h tag under tag 7F2B_h in the Key Creation Request ([Table 11-5](#))

If all the above conditions are successfully verified, then the receiver should accept the share unless other conditions which are outside of the CCC specification apply.

The mailbox size of the relay server shall be large enough to hold all certificates that need to be transmitted in the Key Creation Request.

The key configuration data fields ([Table 11-20](#)) in ASN.1 and TLV format shall be present because the receiver device may not support the key configuration in TLV format. In this case, the receiver shall ignore the key configuration in TLV format, if it is present. The key configuration data fields in ASN.1 format are wrapped under tag 7F30_h, while key configuration data fields in TLV format are wrapped under tag 7F3C_h to avoid ambiguity at the receiver devices that do not support the key configuration data fields in TLV format.

Endpoint configuration data fields shall be provided with tag 7F27_h receiver devices that only support [\[41\]](#) and with Tag 7F2C_h for devices that support this Digital Key specification version.

Tag 60_h groups data elements in endpoint configuration data corresponding to Mb_xVer value 80_h. This allows the vehicle to provide endpoint configuration data corresponding to devices supporting only [41] in a backward compatible manner.

Listing 11-2: Sharing Invitation

```
1  input: user action
2  output: key_creation_request
3  begin
4    generate a key_configuration field as per Table 11-20, sender might be prompted with a specific UI for this action.
5    key_configuration.updateCounter ← 0
6
11   generate an endpoint_configuration field as described in Table 15-13 with
12     endpoint_configuration ← copy sender's endpoint_configuration
13     endpoint_configuration ← remove endpoint_identifier and instance_CA_identifier fields
14     endpoint_configuration.option_group_1.bit4 ← 1
15     endpoint_configuration.option_group_1.bit5 ← 1
16     endpoint_configuration.endpoint_identifier ← see naming rules for shared key (see Section 4.2)
17
18   if confidential_mailbox_size field present in sender's endpoint configuration
19     endpoint_configuration.confidential_mailbox_size ← 1 × IMMOBILIZER_TOKEN_LENGTH
22
23   generate key_creation_request as per Table 11-5 using key_configuration, endpoint_configuration,
24     ROUTING_INFORMATION (, receiver sharing configuration)
25
26   send key_creation_request to receiver over sender-receiver-com
27 end
```

11.4.1.1 Versioning

If non-backward compatible changes in the key creation request (Table 11-5) are necessary, then a new tag (e.g., 7F33_h) shall be defined and both data structures shall be sent to the receiver device, which can then pick the tag it is able to process.

Related applet commands:

- CREATE ENDPOINT (friend)

For forward compatibility, the receiver device shall accept other tags from the sender device under 7F31_h in the Key Creation Request. If not known, these tags shall be ignored by the receiver device. In future versions, the sender device may send other tags.

If Tag 54_h in Table 11-5 is recognized, then the receiver device shall determine a commonly supported key sharing version (OD-FD-KS), send its supported version numbers back in the Key Signing Request (see section 11.4.2) and continue the key sharing process according to the determined version number. If the tag is not recognized, the receiver device shall behave as prior to version introduction.

The endpoint configuration data shall contain all data elements from older versions to be backward compatible. New tag shall be used if format change or extension is needed (Table 11-19).

The receiver device shall reject the sharing if the ROUTING_INFORMATION format cannot be handled by the device OEM server, which processes the data. This data format is determined before the sharing starts, so its version or format cannot be changed.

The receiver device shall reject the sharing or ask the user to update if Digital Key applet transaction (V-D-TX) are not compatible with the receiver applet protocol version. This version list is determined by the vehicle, the device needs to find a matching version from that list. The receiver device shall reject the sharing or ask the user to update if any version of the set of vehicle version lists in tag 5E_h are not compatible with the supported receiver BT versions. Note that tags 4A_h and 4B_h are not likely to change their format. If this becomes necessary, new tags shall be defined and sent in addition.

11.4.2 Steps 6 through 10 (Receiver): Key Signing Request

The receiver device creates an endpoint using the provided endpoint_configuration, then transfers to the sender a Key Signing Request containing the endpoint certificate along with the associated certificate chain. The receiver device uses the list of authorized_PK provided by the sender in the endpoint_configuration field to produce a certificate chain starting from one of these public keys. The receiver device shall persist the authorized_PK list in the receiver endpoint as otherwise re-sharing is not possible.

Table 11-6: Key Signing Request

Tag	Length (bytes)	Description	Field is	Domain Version
7F36 _h	variable	Receiver Key Signing Request		OD-FD-KS
		external CA Certificate container as per Table 11-7	mandatory	V-OD-FW
		instance CA Certificate container as per Table 11-8	mandatory	V-OD-FW
		endpoint certificate container as per Table 11-9 signed by instance CA private key	mandatory	V-OD-FW
		endpoint encryption key attestation as described in Table 15-49 signed by endpoint private key	Conditional. Present only if immobilizer token is required	V-OD-FW
	55 _h	2 x n supported OD-FD-KS compatibility versions of receiver device (ver.high ver.low), where $n \geq 16$ for receivers and $n \leq 16$ for transmitters. The agreed version is first in the list.	mandatory	OD-FD-KS
	5F3F _h	variable receiver-entered device PIN (F_PIN)	Conditional. Required if tag 9F17 _h has been sent in key creation request and activationOption = devicePIN	OD-FD-KS

Tag	Length (bytes)	Description	Field is	Domain Version
			(genericSharingData)	

1

2

Table 11-7: External CA Certificate Container

Tag	Length (bytes)	Description	Field is
7F20 _h	variable		
	external CA certificate as per Listing 15-13 verifiable by an authorized_PK		mandatory

3

Table 11-8: Instance CA Certificate Container

Tag	Length (bytes)	Description	Field is
7F22 _h	variable		
	instance CA Certificate as per Listing 15-16 signed by external CA private key		mandatory

4

Table 11-9: Endpoint Certificate Container

Tag	Length (bytes)	Description	Field is
7F24 _h	variable		
	endpoint certificate as Listing 15-5 signed by instance CA private key		mandatory

5

Listing 11-3: Key Creation Processing

1	input: key_creation_request
2	output: receiver_key_signing_request
3	begin
4	extract key_configuration, endpoint_configuration from key_creation_request
5	check the applet installed on platform supports at least one of the protocol versions present in DIGITAL_KEY_APPLET_PROTOCOL_VERSIONS
6	check proposed entitlements received from sender comply with receiver's policy,
7	receiver may be prompted with a specific UI for this action.
8	receiver may decline sharing and stop the sharing process at this point by sending sharingReceiverCancel (see Table 11-4).
9	check endpoint_configuration complies with receiver's policy
10	check ability to produce a certificate chain starting from one of the authorized_PK
11	present in endpoint_configuration
12	
13	execute framework.getInstance as described in Section 15.4.1.3
14	output: instance
15	
16	generate and add endpoint_identifier, instance_CA_identifier to endpoint_configuration as
17	per receiver device's local policy
18	execute instance.createEndpoint as described in Section 15.4.1.7
19	input: endpoint_configuration,
20	bool_onlineCertificate = false
21	output: endpoint
22	
23	execute endpoint.getCertificate as described in Section 15.4.1.12
24	input: bool_onlineCertificate = false
25	output: endpoint_certificate
26	
27	if tag DA _h is not present, empty, or present in key_creation_request and set to 00 _h

```

28
29     execute endpoint.createEncryptionKey as described in Section 15.4.1.15
30     output: encryption_key_attestation
31
32     generate receiver_key_signing_request as per Table 11-6
33     using external_ca_certificate,
34         instance_ca_certificate,
35         endpoint_certificate,
36         (encryption_key_attestation)
37
38     else
39         generate friend_key_signing_request as per Table 11-6
40         using external_ca_certificate,
41             instance_ca_certificate,
42             endpoint_certificate
43
44     send receiver_key_signing_request to sender over the receiver device to sender device link
45 end

```

1 11.4.2.1 Versioning

2 Related applet commands:

3 - AUTHORIZE ENDPOINT (sender)

4 The receiver device generates the certificates and attestations in the key signing request ([Table](#)
5 [11-6](#)) using the highest version that is compliant with the agreed key sharing version (OD-FD-
6 KS).

7 It then sends the key signing request with the added list of supported receiver versions. The
8 selected, commonly supported version is first in the list.

9 11.4.3 Steps 11 through 13 (Sender): Generate Import Request

10 The sender generates an attestation over the receiver's public key. The sender sends an Import
11 Request to the receiver.

12 The sender shall generate a sharing method attestation and include it into the import request
13 unless agreed otherwise with the vehicle OEM.

14 The sender device shall provide its own key_attestation package to the receiver, if it has not yet
15 been consumed by the vehicle (to be stored in KeyAtt, see section [4.3.1](#)) as well as
16 key_attestation packaged it has received itself from the device that has shared the key to the
17 sender device, and so on (see [Table 11-11](#)).

18 *Table 11-10: Sharing Method Attestation Arbitrary Data to be Hashed and Signed*

Tag	Length (bytes)	Description	Field is
61 _h	variable	Tuple of used sharing method provider identifier and used sharing method group identifier	conditional
	40 _h 2	Sharing method provider identifier as defined in XXX (e.g., 0x0000 = CCC, 0x0001 Apple, 0x0002 = Google, 0x0003 = BMW...)	mandatory
	41 _h variable	Sharing method group identifier defined by the sharing method provider	mandatory
51 _h	20	Receiver key_identifier, SHA-1 hash of the value of the BIT STRING subjectPublicKey	mandatory

1

Table 11-11: Import Request

Tag	Length (bytes)	Description	Field is	Domain Version
7F32 _h	variable	Mailbox Import Request		OD-FD-KS
	key_attestation as described in Table 15-64 (containing entitlement data as per Table 11-12)		mandatory	V-OD-FW
30 _h	variable	Next upstream key_attestation (tag 7F35 _h) created for sender device. Present, if not yet provided to the vehicle	conditional	OD-FD-KS
31 _h	variable	Next upstream key_attestation (after 30 _h) created for sender device of sender device. Present, if not yet provided to the vehicle	conditional	OD-FD-KS
32 _h	variable	... up to MAX_KEY_ATT_COUNT next upstream key_attestation	conditional	OD-FD-KS
The mailbox_mapping table reference by tag 7F4D _h as described in Table 5-13			mandatory	OD-FD-KS
4A _h	variable	Encrypted confidential mailbox data	Conditional. Present only if immobilizer tokens are required	OD-FD-KS
97 _h	65	Encryption public key of the owner prepended by 04 _h	Conditional. Present only if immobilizer tokens are required	OD-FD-KS
7F49 _h	variable	This tag provides configuration data for the supported radios. See Table 19-90	Conditional. Shall be present if the vehicle supports WCC2/WCC3	V-OD-FW
7F2D _h	variable	SharingMethodAttestation Arbitrary data attestation (as per Table 15-65) with sender key using arbitrary data = SHA-256 hash value of Table 11-10	optional	
7F22 _h	variable	Newly issued sender Instance CA Certificate [E] in encrypted data container as per Table 11-7 and Table 11-20 . The intended recipient is the Vehicle OEM Server	Conditional. Required if Tag DB _h in Table 6-3 is present and the value is not zero	
7F21 _h	variable	Content of Table 11-10 without tag 51 _h .	Conditional. Present when	

Tag	Length (bytes)	Description	Field is	Domain Version
			7F2D _h is present.	

Table 11-12: Entitlement Data

Tag	Length (bytes)	Description	Field is	Domain Version
Key configuration as per Table 11-20 (including wrapping tag)			mandatory	Informative
D7 _h	1	Second Factor authentication Required (00 _h =not required OR Online Sharing PIN Flow (Section 11.5) already successfully completed, 01 _h =required for non-approved sharing methods, other=RFU)	mandatory	V-OD-FW
4D _h	8	vehicle_identifier ⁶	mandatory	V-OD-FW
82 _h	1-8	Sender key_slot_identifier, used to identify the sender key for signature verification	mandatory	V-OD-FW

The receiver device obtains slot identifier through key tracking response ([Table 11-19](#)). The sender does not know the slot identifier while sending Import Request (see [Table 11-11](#)), therefore tag 4E_h is not present in Import Request in this Digital Key specification. In this Digital Key specification, the slot identifier is assigned online by vehicle server in Key Tracking Response (see [Table 11-19](#))

Note: In [\[41\]](#), Import Request contains tag 4E_h to identify the slot identifier in case slot identifier is assigned by the vehicle.

Sender key_slot_identifier is added to allow the vehicle to determine the signer key of the attestation package provided to the vehicle. Here the sender device shall add its own slot identifier value.

Listing 11-4: Import Request

```

1  input: friend_key_signing_request
2
3  output: import_request
4  begin
5      extract from receiver_key_signing_request
6          externalCACertificate,
7          instanceCACertificate,
8          endpointCertificate,
9          (encryptionKeyAttestation)
10
11     retrieve key_configuration sent in step1
12     verify endpointCertificate has been created as per step1 and step2 rules, otherwise abort procedure
13
14     generate an entitlement_data field as per Table 11-12
15     If key_slot present in key_creation_request sent in step1
16         entitlement_data.slotIdentifier ← key_slot
17         entitlement_data.vehicleIdentifier ← vehicle_identifier

```

⁶ In prior versions of the Digital Key specifications, vehicle_identifier was a conditional field. Since Tag DA_h is now mandatory with allowable values of 01_h and 03_h only, this field is now mandatory.

```
17  entitlement_data.keyConfiguration ← key_configuration
18
19  if second_factor_authentication_required by Vehicle OEM policy or device policy (see Section 11.2.1.2)
20    entitlement_data.second_factor_authentication_required ← true
21  else
22    Entitlement_data.second_factor_authentication_required ← false
23
24  execute framework.getInstance as per Section 15.4.1.3
25    output: instance
26
27  execute instance.getEndpoint as per Section 15.4.1.9
28    output: endpoint
29
30
31  execute endpoint.authorize as per Section 15.4.1.14
32    input:
33      u16_offset = immo_token_extraction_offset
34      u16_length = immo_token_extraction_length
35      bytes_arbitraryData = entitlement_data
36      bytes_externalCACertificate = externalCACertificate
37      bytes_instanceCACertificate = instanceCACertificate
38      (bytes_encryptionKeyAttestation = encryptionKeyAttestation)
39      bytes_endpointCreationCertificate = endpointCertificate
40    output: bytes_attestation(, bytes_encryptedData, bytes_encSenderePK)
41
42  prepare import_request buffer as per Table 11-11 using
43    bytes_attestation,
44    (bytes_encryptedData),
45    (bytes_encSenderePK)
46
47  send import_request to receiver over the sender device to receiver device link
48  end
```

1 11.4.3.1 Versioning

2 Related applet commands:

- 3 - CREATE ENCRYPTION KEY (sender)
- 4 - SET CONFIDENTIAL DATA (receiver)

5 The import request includes data from different sources.

6 The key attestation format has been pre-agreed between owner device and vehicle (V-OD-FW).

7 The key attestation package and the mailbox layout shall be created in the format defined by the
8 V-OD-FW version supported by the receiver. To achieve this, each OD-FD-KS version
9 represents exactly one version of the key attestation package. The receiver device matches the
10 list of V-OD-FW versions provided by the vehicle and sent by the sender in key creation request
11 (see Table 11-5), determines the highest commonly supported version and responds to the sender
12 with the appropriate OD-FD-KS version,

13 The confidential mailbox data format and encryption public key is also pre-agreed by V-OD-FW.
14 The encryption algorithm is defined by the SET CONFIDENTIAL DATA API and shall be
15 considered when determining by the OD-FD-KS version.

16 The configuration data for the supported radios is based on Table 19-90, which supports adding
17 new tags for backward compatibility.

11.4.4 Steps 14 and 15 (Receiver): Endpoint Data Import

The key_attestation provided by the sender or the Vehicle OEM Server, shall be written into the receiver's private mailbox, and optionally the immobilizer token may be written into the receiver's confidential mailbox. It is read by the vehicle at the first transaction of the shared key and used to verify that the key is authorized by the sender and (optionally) by the key tracking server.

The receiver's private mailbox has the same mapping as the sender mailbox. The receiver's confidential mailbox contains zero or one immobilizer token. The key slot identifier shall only be included by the receiver device because the slot identifiers are retrieved online.

The attestation package version relates to the attestation data fields structure as per the authorize endpoint command (Table 15-24). It covers all fields of this data structure (see Table 11-13).

Data elements with tags 41_h, 92_h, 5A_g, and 78_h are included in the sender signature (Tag 9E_h). Data elements with tags 4E_h, 4F_h, and 48_h are included in the Key Tracking Response (Tag 45_h). (See Table 11-13).

The attestation package structure and the content of the data fields (e.g., entitlements data format) is depending on the agreed V-OD-FW domain version. As different devices may support different Digital Key specification versions, the vehicle must know the format of an attestation package.

The field attestation packet identifies the following changes in the attestation package related to V-OD-FW 0300_h:

- Format change of entitlements data from ASN.1 to TLV
- New data element in the entitlement data (sender key slot identifier)
- New data elements in the vehicle OEM server-provided part of the attestation package (groupIdentifier, deviceType).

Vehicles supporting only V-OD-FW = 0100_h will never be presented the new attestation package format. Vehicles supporting V-OD-FW = 0100_h and V-OD-FW = 0300_h shall differentiate between two formats by checking the value of the attestation package version (tag 41_h).

The public key of the receiver endpoint (Tag 5A_h) can be presented in normal or compressed format defined by the option_group setting (see section 15.3.2.4).

The groupIdentifier is assigned by the vehicle OEM to each key based on the Account Info Hash provided during Key Tracking Response. The value is at discretion of the vehicle OEM and shall be unique per vehicle. Devices with the same Account Info Hash shall be assigned the same GroupIdentifier. The deviceType is provided in the Key Tracking Response.

V1 keys on the same account shall be assigned the same groupIdentifier as for V3 keys. The vehicle OEM server can determine the associated V1 keys per account by comparing with the accountIdHash of V1 and V3 keys, as devices that support V3 keys are still providing their accountIdHash in trackKey() request API calls. V1 keys shall be added to sharedKeysData in sharedAccount in the same way as V3 keys.

Note: During partial migration (Section 2.10.5.2) only, the groupIdentifier is created using accountIdHash instead of accountInfoHash.

Table 11-13: Attestation Package

Tag	Length (bytes)	Description	Field is	Domain Version
7F35 _h	variable	Attestation Package		V-OD-FW
41 _h	1	Attestation package version = 03 _h	mandatory	V-OD-FW
92 _h	8	Random as described in Table 15-24	mandatory	V-OD-FW

Tag		Length (bytes)	Description	Field is	Domain Version
	5A _h	65 or 33	Public key from receiver endpoint	mandatory	V-OD-FW
	78 _h	variable	Entitlements data as per Table 11-12	mandatory	V-OD-FW
	9E _h	64	Signature with sender endpoint private key over Section Table 15-24 as per Listing 11-2	mandatory	V-OD-FW
	4E _h	1-8	key_slot_identifier – only present if key slot identifiers are retrieved online (i.e., tag DA _h in Table 11-5 is present and set to 01 _h or 03 _h)	conditional	V-OD-FW
	4F _h	2	groupIdentifier	conditional	V-OD-FW
	48 _h	1	deviceType: 01 _h = phone, 02 _h = watch, 40 _h = plastic card, 50 _h = key fob, 80 _h = delegate server	conditional	V-OD-FW
	45 _h	variable	ktsSignature delivered in key Tracking Response. Maximum Length = 320 bytes	mandatory	V-OD-FW

1 *Listing 11-5: Receiver, Endpoint Data Import Processing*

```

1 input: import_request (Table 11-11)
2 output: n/a
3 begin
4   extract tags content from import_request as separate buffers:
5     key_attestation
6     attestation_package(s)
7     mailbox_mapping
8     (encrypted_confidential_mailbox_data),
9     (encryption_public_key),
10
11   extract receiver private mailbox mapping offsets from mailbox_mapping
12
13   execute framework.getInstance as per Section 15.4.1.3
14   output: instance
15
16   execute instance.getEndpoint as per Section 15.4.1.9
17   output: endpoint
18
19   if key tracking required as per tag DBn from Table 11-5 or
20   online attestation delivery is supported as per tag DCn from Table 11-5
21   if tag DAn is present in key_creation_request and set to 01n or 04n (for non-owner device only)
22     execute endpoint.createEncryptionKey as described in Section 15.4.1.15
23     output: encryption_key_attestation
24     Execute Listing 11-7 (using encryption_key_attestation) from Section 11.6 to retrieve response
25
26   prepare attestationPackage buffer as per Table 11-13 using contents of:
27     key_attestation,
28     (Key Tracking Receipt)
29
30   execute endpoint.setPrivateData as per Section 15.4.1.18

```

```
30  input: offset = KEY_ATT_OFFSET, data = attestationPackage(s)
    input: offset = SLOT_ID_OFFSET, data = slotIdentifiers
31
32  setPrivateMailboxBit(endpoint, DEVICE_KEY_ATT_BIT, SIG_BMP_OFFSET) as per Listing 11-14
33
34  from that point a notification indicating success of the online attestation delivery can be received from server
35  on reception of such notification the following cleanup procedure is executed:
36  clearPrivateMailboxBit(endpoint, DEVICE_KEY_ATT_BIT, SIG_BMP_OFFSET) as per Listing 11-15
37  execute endpoint.setPrivateData as per Section 15.4.1.18
38  input: offset = KEY_ATT_OFFSET, data = zeroes
39
40  if tag 4Bh was received in Step 1
41  execute endpoint.setParameters with the following parameters:
42  input: offset_private = offset extracted from tag 4Bh,
43  length_private = length extracted from tag 4Bh
44
45  if tag DAh from Table 11-5 was present in Step 1, and is empty or equal to 00h or 01h
46  execute endpoint.setConfidentialData as per Section 15.4.1.16
47  input:
48  bytes_encsenderepk = encryption_public_key
49  bytes_encdata = encrypted_confidential_mailbox_data
50  u16_offset = 0
51  u16_length = IMMOBILIZER_TOKEN_LENGTH
52
53  if tag 4Ah was received in Step 1
54  execute endpoint.setParameters with the following parameters:
55  input: offset_confidential = offset extracted from tag 4Ah,
56  length_confidential = length extracted from tag 4Ah
57  setPrivateMailboxBit(endpoint, 0, SLOT_ID_OFFSET) as per Listing 11-14
58
59  end
```

11.4.5 Step 18 (Receiver): Track Key

The receiver device in Step 18 shall send trackKey (see Section [17.7.3.2](#)) to the Vehicle OEM Server via the receiver Device OEM Server to obtain a key tracking receipt. Note that step 18 is shown for illustration purposes only.

If slot identifier is retrieved online, the receiver device shall keep the key disabled on the NFC and BLE interfaces until a valid slot identifier is available. The key shall be usable on the NFC and BLE interfaces only after a successful trackKey() Response(see Section [17.7.3.3](#)) containing the slot identifier.

The Vehicle OEM server shall verify the freshness and validity of the owner/sender Instance Certificate during key tracking if tag DB_h was present with a non-zero value in the owner/sender trackKey request (see Section [6.3.4.3](#)). In the case of failure, the Vehicle OEM Server shall respond in trackKey() Response with Sub Status Code equal to 50118, i.e., Invalid or expired Owner Instance CA Certificate, as described in Section [17.11.3](#). The Vehicle OEM Server shall also send an eventNotification SHARED_KEY_REJECTED to the owner Device OEM Server.

11.4.6 Step 19 (Vehicle OEM Server): Track Key Response

In step 19, the Vehicle OEM Server shall send trackKey() Response (see Section [17.7.3.3](#)) to the receiver Device OEM Server. Note that step 19 is shown for illustration purposes only.

11.4.7 (Shared Key): First Transaction

This section describes the operations occurring between receiver device and vehicle during the first presentation of a shared key endpoint on which operations described in Section 11.4.3 have been successfully completed.

Listing 11-6: First New Key Transaction

```
begin
vehicle and device execute the standard transaction as described in Section 7 until
  AUTH1 response from the device
device AUTH1 response contains: (as described in Section 15.3.2.10)
  the key_slot
  the endpoint_signature
  (confidential_mailbox_subset: offset = 0, length = IMMOBILIZER_TOKEN_LENGTH)
  private_mailbox_subset: offset = SIG_BMP_OFFSET, length = 1
vehicle extracts signaling_bitmap ← private_mailbox_subset[0]
if immobilizer token required by vehicle
  vehicle extracts immobilizer_token ← confidential_mailbox_subset[0]
  vehicle verifies if DEVICE_KEY_ATT_BIT in signaling_bitmap is not set
  no attestation package is present, exit and process as regular transaction
  vehicle verifies if DEVICE_SLOT_ID_BIT in signaling_bitmap is not set
  error
  vehicle reads SlotIdentLookup data structure
  vehicle searches for the first slot identifier that does not have a PK associated in the vehicle and assigns index i depending on
  its position in the SlotIdentLookup data structure (i = 1, for the first slot identifier)
  If no empty slot identifier is found
  end of flow
  vehicle reads attestation package i, starting at (i-1)x MAX_KEY_ATT_LENGTH
  vehicle verifies attestation package using public key from sender key_slot_identifier in Tag 82n in Entitlements data (see Table
  11-12)
  if verification is not successful
  error
  if vehicle signature verification of current attestation package is successful
  vehicle verifies key has not been registered yet, otherwise abort
  vehicle verifies value of slot identifier is valid, otherwise abort (conditional, this step is not performed if slot identifier is retrieved
  online)
  vehicle verifies all elements of the attestation package are compliant with its policy, otherwise abort
  vehicle registers endpoint.PK and entitlements
else
  abort procedure
if unprocess values remaining in SlotIdentLookup data structure
  goto line 17
  vehicle verifies the endpoint_signature received in AUTH1
  vehicle clears DEVICE_KEY_ATT_BIT in signaling_bitmap
  vehicle clears the attestation package from device private mailbox using the one or multiple EXCHANGE commands
  using value from attestationPackage.second_factor_authentication_required and information present in key tracking receipt
  the vehicle verifies the second factor authentication mechanism
End
```

If the vehicle detects after reading the SlotIdentLookup data structure that it has to read more than one attestation package from the private mailbox, then the vehicle shall send a control flow command “long processing time (UI indication)” (see Table 15-46). The vehicle might require additional activation of a shared key to start the engine. In this case the activation options have been provided to the sender in the key tracking request or via event notification. The sender has provided the list of activation options to the receiver during key sharing.

11.5 Online Sharing PIN Flow

11.5.1 Concept

All devices shall support online sharing PIN that is generated on the sender device, entered on the receiver device and verified by the vehicle OEM server. OSP (or another activation options) must be used if required by the vehicle OEM. Device PIN as described in Section [11.6](#) should be offered as an alternative second factor that can be activated by the sender during the sharing workflow, only if the vehicle OEM does not mandate any activation option.

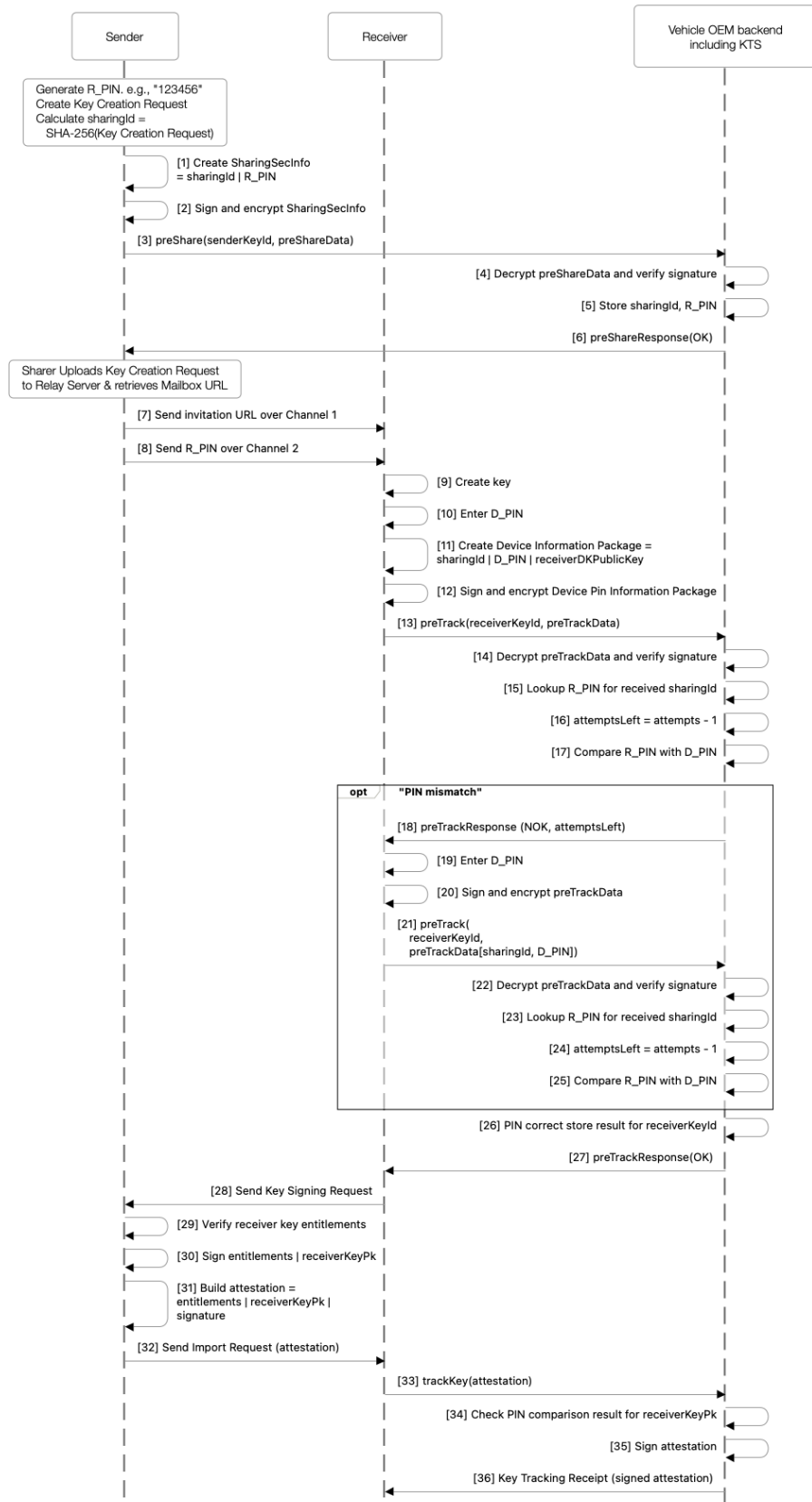
Device UIs shall guide users to transfer the online sharing PIN over a second channel and guard against the user unintentionally re-using the same channel used to communicate the sharing invitation.

To ensure a good user experience, it is recommended that multiple activation options that require user interface interaction should not be activated at the same time. For example, online sharing and device PIN.

Please see Section [11.2](#) entitled “Sharing Principles” about vehicle OEM-defined activation options and device PIN policies.

1

Figure 11-2: Online Sharing PIN Flow



2

11.5.2 Online Sharing PIN Flow: Steps 1 through 8:

The sender device generates the Key Creation Request ([Table 11-5](#)) and the reference online sharing PIN (R_PIN). It sets the number of online sharing PIN entry attempts (Tag 9F17_h) and length of online sharing PIN (Tag D8_h) to values received from vehicle OEM in the trackKey() response. See sharingPasswordLength and maximumOnlineSharingPINAttempts in [Table 17-51](#).

The sharingId (same as included in genericSharingData, Section [11.3.3](#)) is generated by calculating a SHA-256 hash of the Key Creation Request ([Table 11-5](#)). The sharingId is a unique identifier for that sharing session.

The Sharing Sec Info (7F39_h) contains the R_PIN and sharingId. The Sharing Sec Info is signed by the sender and resulting Signed Sharing Sec Info (7F41_h) is encrypted with the encryption key of the Vehicle OEM Server. The encrypted data is sent as preShareData via the preShare() API ([Section 17.7.6](#)) to the Vehicle OEM Server. Vehicle OEM server decrypts the data, verifies that the signature is correct and from an eligible Sender, stores the data and acknowledges the reception. .

11.5.3 Online Sharing PIN Flow: Steps 9 through 13:

The receiver claims the sharing URL and creates the Digital Key. The receiver extracts the R_PIN value from the second channel. R_PIN may be delivered to the receiver over a voice call. This value is now named D_PIN. The receiver enters the D_PIN into the receiver device UI and creates the Online Sharing Pin Information Package (7F42_h), [Table 11-14](#). The package includes the sharingId (as received from genericSharingData, ref.: Chapter [11.3.3](#)), the D_PIN and the public key of the Digital Key of the receiver (receiverDkPublicKey). The receiverDkPublicKey is later used by the vehicle OEM server to verify preTrack and trackKey calls were both sent by the same device. The Online Sharing Pin Information Package is signed by the receiver and resulting Signed Online Sharing Pin Information Package (7F43_h) is encrypted with the encryption key of the Vehicle OEM Server. The encrypted data is sent as vehicleData via preTrack() to the Vehicle OEM Server.

Table 11-14: Online Sharing PIN Information Package

Tag	SubTag	Length (bytes)	Description	Field is	Domain Version
7F42 _h		variable	Online Sharing Pin Information Package	mandatory	
	5F53 _h	64	sharingId	mandatory	
	5F3F _h	variable	Receiver-entered Online Sharing PIN (D_PIN)	mandatory	
	50 _h	65	receiverDkPublicKey (prepended by 04 _h)	mandatory	

Table 11-15: Unencrypted Signed Online Sharing PIN Information Package

Tag	Subtag	Length (bytes)	Description	Field is	Domain Version
7F43 _h		variable	Signed Online Sharing Pin Information Package	mandatory	N/A. Informative Only
content of Table 11-14 (tag 7F42 _h)				mandatory	N/A. Informative Only
SIG-DAT: content of Table 15-58 (Signature Data Fields) with: arbitrary_data = SHA-256 hash value of Table 11-14 (7F42 _h)				mandatory	
	9E _h	64	signature with the private key of the receiver Digital Key over fields from SIG-DAT	mandatory	N/A. Informative Only

11.5.4 Online Sharing PIN Flow: Steps 14 through 27: PIN validation

Upon receiving the preTrack request, the Vehicle OEM server decrements the number of attempts left for PIN input. The Vehicle OEM server looks up the R_PIN corresponding to the sharingId and compares the received D_PIN with the R_PIN value. If both values are equal, the Vehicle OEM server stores the result for the receiverDkPublicKey and sends a preTrack() Response call with status OK.

If the D_PIN entered by the receiver does not match, the vehicle OEM backend sends a preTrack() Response with status OK along with the attemptsLeft to the receiver device.

The vehicle server shall ensure that this key sharing is not activated using any other activationOptions, if the PIN validation fails and the attemptsLeft is zero.

11.5.5 Online Sharing PIN Flow: Steps 28 to 36: Key Sharing finalization

After PIN verification was confirmed by the Vehicle OEM server, the key sharing flow continues on the receiver device. When the receiver device finally reaches out to the KTS to track the shared key, the KTS performs the following verifications:

1. The receiverDkPublicKey (taken from Endpoint Certificate as received in encrypted keyData container) is the same as the one received in previous preTrack call and
2. Ensures that PIN verification for the receiverDkPublicKey was successful.

KTS shall sign the trackKey request only if both verifications are successful, else the tracking is declined. The Vehicle OEM server shall ensure that after successful OSP verification, no further activation options are enforced, i.e., Table 11-12, Tag D7_h is either absent or set to 00_h.

11.6 Device PIN

11.6.1 Concept

All devices shall support Device PIN that is generated on the owner device and entered on the friend device. Device PIN could be activated by the owner during the sharing workflow if no other vehicle OEM-defined activation option is used. Device UIs shall guide users to transfer the device PIN over a second channel and guard against the user unintentionally re-using the same channel used to communicate the sharing invitation. The friend then types the PIN into the friend

device UI to continue the sharing workflow. The owner device then verifies that the correct PIN was entered into the friend device UI.

The owner device policy determines whether a sharing method requires the device PIN. The owner device policy is determined by the owner device OEM and is out of scope. To ensure a good user experience, it is recommended that multiple activation options that require user interface interaction should not be activated at the same time. For example, online sharing PIN (controlled by vehicle OEM) and device PIN (controlled by device OEM)

Please see Section [11.2](#) entitled “Sharing Principles” about vehicle OEM-defined activation option and device PIN policies.

11.6.1.1 Basic Flow

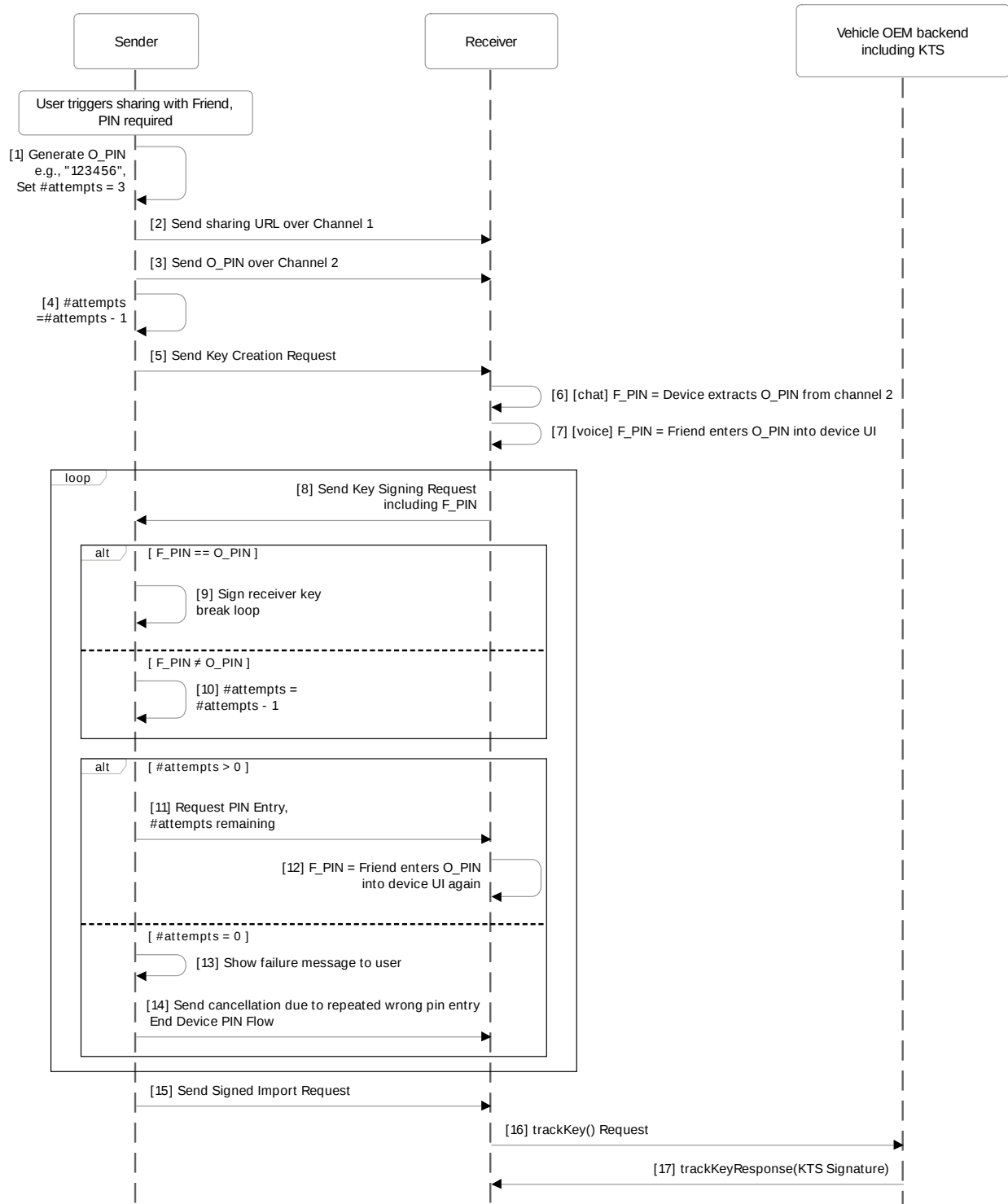
The owner device generates the reference device PIN (O_PIN). It sets the number of device PIN entry attempts to a device OEM-defined value, considering the *pinLength* as described in Section [11.3.3](#).

The owner then sends the O_PIN over a second channel to the friend. This step shall be guided by the device UI to ensure that the device PIN is transmitted while complying with restrictions in [11.6.1](#). The owner device decrements the number of attempts by one.

The owner device sends the initial number of attempts in the key creation request via tag 9F17_h. The presence of this tag indicates that the owner has requested the entry of device PIN to the friend device.

1

Figure 11-3: Device PIN Flow



2

3 For a more focused view, [Figure 11-3](#) does not show the device OEM server.

4 The friend extracts the O_PIN value from the second channel or receives it over a voice call.

5 This value is now named F_PIN. The friend enters the F_PIN into the friend device UI and the

6 friend device adds the F_PIN to the key signing request.

The owner device compares the received F_PIN with the O_PIN value. If both values are equal, then the owner device signs the friend key and sends the import request. If the device PIN values are different, then the owner device sends a sharingPinReEntryRequest message (as per [Table 11-16](#)) to the friend. The sharingData structure is defined in [Table 11-2](#).

Table 11-16: Device PIN Entry Request

Tag	Length (bytes)	Description	Field is
9F17 _h	1	Number of device PIN entry attempts	mandatory

The friend device allows the friend to enter the F_PIN again and sends the input back by sending a sharingPinReEntryValue message (as per [Table 11-4](#)) to the owner. The content is identical with the key signing request in [Table 11-6](#), but the sharingDataType indicates that this is a repeated device PIN entry attempt.

The owner compares again F_PIN with O_PIN and, if equal, decides to sign or, if different, to allow a new attempt to the friend device until either a correct F_PIN has been provided or the number of remaining attempts is 0.

If the number of remaining attempts is 0, then the owner sends a cancel message ([Table 11-4](#), *sharingOwnerCancel*) indicating that no valid device PIN has been received in the defined number of attempts.

If the friend device does not send back tag 5F3F_h (F_PIN) in the signing request, then the friend device does not support the device PIN feature. The owner device policy for this case is defined by the device OEM. Options could be to reject sharing to this device or propose to the owner to share without device PIN, providing some security education.

11.7 Key Tracking and Online Attestation Delivery

The attestation package contains a signature computed over elements from [Table 11-13](#). This signature shall be verified by the vehicle before accepting a new key. In that case, the key sharing flow requires an additional interaction with a Vehicle OEM Server.

If supported by the Vehicle OEM, the attestation package provided may also be sent directly to the vehicle via the Vehicle OEM telematic link (if the vehicle is reachable) to speed up the First New Key Transaction. This feature is called Online Attestation Delivery.

The cryptographic algorithm and final data elements used by the KTS to produce the receipt are specific to each Vehicle OEM and are out of scope of this specification.

Section [17.7.3](#) (API – Track Key) describes how the data elements highlighted in the paragraphs above are packaged and sent to the Vehicle OEM. This function assumes the Vehicle OEM Server possesses the owner’s public key to verify the signature of the incoming packets.

Before the key tracking has taken place, a termination request may be received by the vehicle OEM server. In such a case, a key tracking receipt shall not be sent in response.

The Vehicle OEM Server may store the receiver’s endpoint public key and the Instance CA public key. The Device OEM may store an immutable unique device identifier for a given Instance CA public key, such that if the two parties cooperate, it is possible to link an endpoint public key with an immutable unique device identifier for legal and investigational purposes.

1

Table 11-17: Key Tracking and Online Attestation Delivery Request

Tag	Length (bytes)	Description	Field is	Domain Version
7F38 _h	variable	Receiver Key Tracking and Online Attestation Delivery Request		V-OD-FW, D-VS
		Attestation package without tags 4E _h , 4F _h , 48 _h , and 45 _h (ktsSignature). See Table 11-13	mandatory	V-OD-FW
		Receiver endpoint certificate container signed by instance CA. See Table 11-9	mandatory	V-OD-FW
		Receiver Instance CA Certificate container signed by external CA. See Table 11-8	mandatory	V-OD-FW
		Endpoint encryption key attestation signed by endpoint private key. See Table 15-49	Conditional. Present only if immobilizer token is required	D-VS
5F49 _h	65	Device privacy encryption key (Device.Enc.PK)	mandatory	D-VS
DA _h	Variable (maxlen : 10 bytes)	Device privacy encryption version, Default "ECIES_v1" (ASCII)	mandatory	D-VS
7F2D _h	variable	SharingMethodAttestation Arbitrary data attestation (as per table 15-61) with sender key using arbitrary data = SHA-256 hash value of Table 11-10	Conditional. Receiver device shall provide the attestation if it has been received from sender device	D-VS
7F21 _h	variable	Content of Table 11-10 without tag 51 _h .	Conditional: present when 7F2D _h is present.	D-VS
7F23 _h	variable	Newly issued sender Instance CA Certificate [E] in encrypted data container as per Table 11-18 and Table 11-8	Conditional: Required if Tag 7F22 _h in Table 11-11 is present	
5F53 _h	64	sharingId	Conditional: Present during intra-account sharing or using online sharing PIN for 2 nd factor authentication	
5E _h	32	Account Info Hash	mandatory	D-VS
DB _h	1	Sender Device Instance CA freshness requirement for Key Sharing in hours. No freshness check required if tag is absent or 0.	optional	

2

Table 11-18: Encrypted Data Container for Owner Instance CA Certificate

Tag	Length (bytes)	Description	Field is
DA _h	variable (maxlen: 10 bytes)	Identifier for the algorithm specified in this document. Example: "ECIES_v1"	mandatory
97 _h	variable	This should be from an ephemeral key pair generated for a single message. Format for encoding the sender public key: concat(0x04, x, y) (This will be 65 bytes total for a key generated based on named curve secp256r1 - 1.2.840.10045.3.1.7)	mandatory
46 _h	variable	Recipients' key agreement public key fingerprint. Fingerprint generation algorithm: sha256Hash(toUncompressedRawECPublicKey-Format(recipientKAPublicKey)). SHA-256 hash of the uncompressed key agreement public key (concat(0x04, x, y)) of the receiving system used in ECIES_v1 Key Agreement	mandatory
4A _h	variable	Encrypted data: encrypt(unencryptedData, symmetricEncryptionParams)	mandatory

Table 11-19: Key Tracking Response

Parameter	Length (bytes)	Description
ktsSignature	variable	See Table 17-40
slotIdentifier	1-8	See Table 17-40
confidentialMailboxData	variable	See Table 17-40
kBleOobKey	variable	See Table 17-40
kBleIntroKey	variable	See Table 17-40
groupIdentifier	2	See Table 17-42
deviceType	1	See Table 17-57

All the parameters in the Key Tracking Response from the Key Tracking Server shall not be provided in TLV format. The server API parameters shall be provided within a JSON structure defined in the relevant sections in [Table 11-18](#).

Endpoint encryption key attestation (Tag 7F26_h) shall be included in [Table 11-17](#) when Tag DA_h in tag 7F60_h in [Table 11-5](#) is set to 01_h.

Receiver slot identifier (Tag 4E_h) shall be included in

- 1 Table 11-18 when tag DA_h in Tag 7F60_h in [Table 11-5](#) is set to 01_h or 03_h.
2 Receiver immobilizer token (Tags 4A_h and 97_h) shall be included in
3 Table 11-18 when Tag DA_h in Tag 7F60_h in [Table 11-5](#) is set to 01_h.
4 kBleOobKey shall be included if the agreed OD-FS-KS version is 0300_h or higher. The receiver
5 device shall overwrite the kble_oob received in Import Request, if a kBleOobKey is provided in
6 Key Tracking Response.

7 *Listing 11-7: track key Response Handling*

```
1 input: (encryption_key_attestation)
2 begin
3   device creates receiver device to Device OEM Server link
4   device prepares message as per Table 11-17 using (encryption_key_attestation)
5   device sends message over the Device OEM Server link to Vehicle OEM Server link as per Section 17 (API – trackKey)
6   Vehicle OEM Server conditionally (as described in 6.3.4.3) the owner instance CA certificate contained in the Key Tracking
   Request
7   Vehicle OEM Server verifies owner signature contained in attestation package
8   Vehicle OEM Server optionally tries to push the attestation package via vehicle telematic link
9   Vehicle OEM Server optionally computes key tracking receipt as per vehicle OEM proprietary specification
10  device includes tag 45h and, if slot identifier is retrieved online 4Eh from Table 11-13 received from server into
   Table 11-13
11  response may be empty if vehicle OEM does not require key tracking receipt
12  device updates slot identifier as per Section 15.3.2.21 SETUP ENDPOINT command (if slot identifier is retrieved online)
13  device writes confidential mailbox using encrypted confidential mailbox data as per Section 15.3.2.20 SET CONFIDENTIAL
   DATA command (if immobilizer token is retrieved online)
14 end
```

8 **11.7.1 Online immobilizer token and slot identifier retrieval**

- 9 In this Digital Key specification, the slot identifier and (optional) immobilizer token shall be
10 retrieved by the receiver device online, together with the Online Attestation package.
11 In this case, the owner shall not send a slot identifier in the Key Creation Request and shall set
12 the tag DA_h (see [Table 11-5](#)) to 01_h to indicate the use of online immobilizer token, or to 03_h to
13 indicate the use of online slot identifier without immobilizer token. After the key signing request,
14 the owner will only use the key identifier to identify the receiver key. The Key Tracking
15 Response includes the immobilizer token and/or the slot identifier which has been assigned to
16 this receiver key by the Vehicle OEM server. The receiver shall store the immobilizer token in
17 the confidential mailbox using the SET CONFIDENTIAL DATA command and/or update the
18 slot identifier using the SETUP ENDPOINT command.

19 **11.7.2 Online BLE Keys retrieval**

- 20 Akin to slot identifier and immobilizer token, BLE keys are required to be sent by the vehicle
21 OEM server if non-owner devices share keys. The vehicle has the possibility to override
22 kble_oob_master derived during SPAKE2+ protocol during owner pairing with a value that is
23 synchronized with the vehicle backend. Tag D5_h is introduced into the wireless configuration
24 structure Tag 7F49_h for this purpose. This is useful if an owner device supporting SiaC shares a
25 Digital Key to a device with older version of Digital Key specification without online BLE Keys
26 support. Vehicle OEMs that support SiaC shall support online BLE keys retrieval with
27 onlineBleKeysRequired boolean in v1/trackKey0.

Multiple shared keys with online BLE keys shall be accepted by the vehicle in a row without the need to connect to internet between sharings.

11.8 Owner Device OEM Server Notification

11.8.1 Vehicle OEM Server: eventNotification

Following successful activation of the shared key([Figure 11-2](#)), the Vehicle OEM Server shall send an eventNotification KEY_INFORMATION_UPDATED (see Section [17.9.1](#)) to all the Device OEM Servers that have visibility on the active shared key. This may occur before or after the first shared key contact with the vehicle. For those keys where the device can manage the key, the flag managementEnabled shall be set to 1.

Example:

Keys/ Devices:

A: owner device.

B: can see all keys but can only manage downstream keys

C: can see and manage only downstream keys.

D: can see and manage only downstream keys.

Sharing flow:

A → B → C

A → D

Event Notifications.

1. A → B: A receives an event notification, SHARED_KEY_ADDED, listing the key B with managementEnabled = TRUE for B
2. B → C: A and B receive an event notification, SHARED_KEY_ADDED, listing the key B with
 - a. A having managementEnabled = TRUE for C and
 - b. B having managementEnabled = TRUE for C (B can delete C)
3. A → D: A and B receive an event notification, SHARED_KEY_ADDED, listing the key D with
 - a. A having managementEnabled = TRUE for D and
 - b. B having managementEnabled = FALSE for D (B cannot delete D)

11.8.2 Owner Device OEM Server: eventNotification() Response

The sender Device OEM Server and all device OEM servers that have received an eventNotification as outlined in Section [11.8.1](#) shall send eventNotification() Response (see Section [17.9.1.3](#)) to the Vehicle OEM Server.

11.9 Key Configuration

A TLV encoded schema defines the Digital Key configuration with available access profiles and their encoding. This schema is described in [Table 11-20](#). Note that prior versions of this specification used ASN.1 formatting for the Digital Key configuration.

Table 11-20: Key Configuration in TLV Format.

Tag	Length (bytes)	Description	Field is	Domain Version
7F3C _h	variable	Key Configuration	Mandatory	V-OD-FW
D0 _h	1	Standard access profiles as defined in Table 11-21 . Maximum length of profile is 1 byte.	Conditional, present if no proprietary profile is present. Either Tag D0 _h or D1 _h shall be present	V-OD-FW
D1 _h	variable	Proprietary access profile	Conditional, present if no standard profile is present. Either Tag D0 _h or D1 _h shall be present	V-OD-FW
D5 _h	2	accountRole (see Table 2-3)	mandatory	V-OD-FW
51 _h	15 or 13	not_before, DER encoded GeneralizedTime (15 bytes in length) or UTCTime (13 bytes in length) as per RFC 5280 [2]	mandatory	V-OD-FW
52 _h	15 or 13	not_after, DER encoded GeneralizedTime (15 bytes in length) or UTCTime (13 bytes in length) as per RFC 5280 [3]	mandatory	V-OD-FW
D3 _h	variable	Friendly name of the receiver key. Limited to 30 characters	mandatory	V-OD-FW

Table 11-21: Standard Access Profiles

Profile	Type	Description
0	Full	Full access and Drive Capabilities
1	Access Only	Only car access, no drive capabilities
2	Access and Drive Restricted	Access and Drive with Restrictions
3	Car Delivery	Car Delivery Profile
4	Valet	Valet Parking Key
5	Vehicle Service (full)	Vehicle Service Key (full drive)
6	Vehicle Service (restricted)	Vehicle Service Key (restricted drive)

Profile	Type	Description
7-- 255	RFU	

Proprietary access profiles can be coded in a vehicle-specific method within tag D1_h.

11.10 Intra-Account Sharing with Secure Receiver Verification

To avoid the need for 2nd factor activation when sharing a key intra-account (e.g., from phone to a wearable with the user is logged in with same device OEM account on both devices) a Device OEM Account Hash comparison method can be used. During Key Tracking, the KTS server stores the Device OEM Account Info Hash of the receiver of the keys. If there is still an activated digital key associated with the Account Info Hash, the KTS Server can allow sharing without the need of a 2nd factor.

11.10.1 Steps 1 to 6: Initiate Sharing

When sharing to a device within the same Device OEM Account, Sender generates the Key Creation Request and an anonymized receiver Account Info Hash. The sharingId (same as included in genericSharingData, Section [11.3.3](#)) is generated as SHA256 of the Key Creation Request. The Sharing Sec Info (7F39_h) contains the receiver AccountInfoHash and sharingId. The Sharing Sec Info is signed by the sender and resulting Signed Sharing Sec Info (7F41_h) is encrypted with the encryption key of the Vehicle OEM Server. The encrypted data is sent as preShareData via the preShare() API to the Vehicle OEM Server. Vehicle OEM server decrypts the data, verifies that the signature is correct and from an eligible Sender, stores the preShareData and acknowledges its reception.

11.10.2 Step 7 and 8: Key Sharing

The Sender performs cross platform key sharing as described in chapter [11](#).

Figure 11-4: Secure Intra Account Sharing

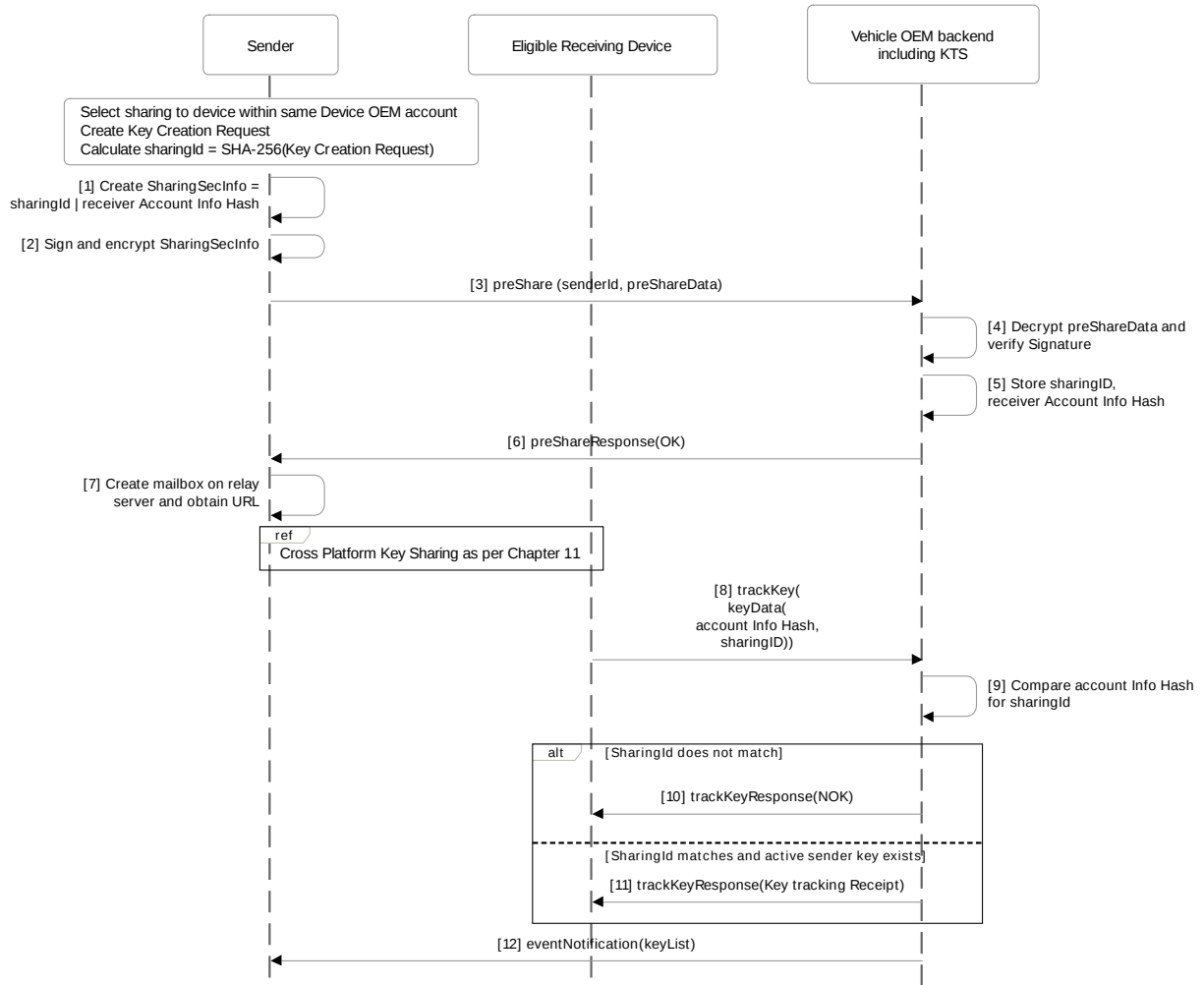


Table 11-22: Sharing Sec Info

Tag	SubTag	Length (bytes)	Description	Field is	Domain Version
7F39 _h		variable	Sharing Sec Info	mandatory	
	5F53 _h	32	sharingId	mandatory	
	5F54 _h	Var	R_PIN	Conditional. Required if online sharing PIN shall be used as 2nd factor for key sharing	
	5F55 _h	var	Receiver accountInfoHash	Conditional. Required if Receiver Account Info Hash shall be used as 2nd factor for key sharing	

Table 11-23: Unencrypted signed SharingSecInfo

Tag	Length (bytes)	Description	Field is	Domain Version
7F41 _h	variable	Signed Sharing Sec Info	mandatory	N/A. Informative Only
contents of Table 11-22 (tag 7F39 _h)			mandatory	N/A. Informative Only
SIG-DAT: content of Table 15-58 (Signature Data Fields) with: arbitrary_data = SHA-256 hash value of Table 11-22 (7F39 _h)			mandatory	
	9E _h 64	signature with the private key of the sender Digital Key over fields from SIG-DAT	mandatory	N/A. Informative Only

11.10.3 Steps 9 to 12: Account Info Hash verification

Upon receiving the trackKey request, the Vehicle OEM compares the receiver Account Info Hash with the Account Info Hash of the trackKey request. For enhanced security the value account_info_hash from receiver endpoint certificate ([Listing 15-3](#)) should be used. If the hash received don't match the tracking will be declined. Otherwise, attestation package will be signed by KTS and forwarded to the receiver.

11.11 Certificate Chain

The goal of the certificate chain is to allow the sender's SE to verify that a receiver's endpoint is authorized to participate in the sharing protocol. This verification is required to make sure that the receiver key pair has actually been generated on an approved SE, and to ensure that any secret data transferred from sender's confidential mailbox to a receiver's confidential mailbox shall be handled as per the rules described in [Section 4.3](#).

Each owner endpoint contains one or several authorized_PK set by the vehicle during the owner pairing process as per [Section 6](#). The authorized_PK list contains, at a minimum, the public key of the Vehicle OEM CA. The authorized_PK list may only contain intermediate CA or root CA but no leaf certificates. The External CA is typically the Device OEM CA used to issue the Instance CA Certificate of Digital Key applet. The Instance CA in turn issues the endpoint certificates.

In this version of the specification, only one single authorized_PK shall be set by the vehicle. The verification of the receiver endpoint is handled through the following verification chain. Each element to the right of the arrow is attested by the element above it:

Authorized CA (Vehicle OEM CA)
 —> External CA (Device OEM CA)
 —> Instance CA
 —> Endpoint
 —> Endpoint Encryption Key

The SBxD/KIS endpoint certificate is akin to the endpoint certificate in [Listing 15-5](#). It contains extension as described in [Listing 11-10](#) in addition to the endpoint certificate extension from [Listing 15-3](#).

The SBxD/KIS intermediate certificate is akin to instance CA certificate in [Listing 15-16](#). It contains the extension as described in Listing 6 instead of the instance CA extension from [Listing 15-14](#).

11.12 Key Classes

It is required that the receiver device can verify the authenticity of the sender device. Therefore, the sender provides their endpoint or SBxD certificate chain to the receiver device in the key creation request.

Given is the following sharing chain: Owner Device → SBFD Delegate Server → Service device

The tag 7F2B_h for the Key Creation Request to SBFD shall contain the following:

```
7F2B xx
  30 xx
    7F20 xx
      Owner external CA certificate [F]
    7F22 xx
      Owner instance CA certificate [E]
    7F24 xx
      Owner endpoint CA certificate [H]
```

Tag 7F2B for the key creation request to the service device shall contain the following:

```
7F2B xx
  30 xx
    7F20 xx
      Owner external CA certificate [F]
    7F22 xx
      Owner instance CA certificate [E]
    7F24 xx
      Owner endpoint CA certificate [H]
  31 xx
    7F44 xx
      SBFD endpoint certificate [T]
```

Table 11-24: SBxD/KIS Intermediate Certificate Container.

Tag	Length (Bytes)	Description	Field is
7F42 _h	variable	SBxD/KIS Intermediate Certificate (see Listing 11-13)	mandatory

Table 11-25: SBxD/KIS Endpoint Certificate Container.

Tag	Length (Bytes)	Description	Field is
7F44 _h	variable	SBxD/KIS Endpoint Certificate (see Listing 11-10)	mandatory

Listing 11-8: SBxD/KIS Endpoint Certificate Extension Data

```
SBxDKisCertificateExtensionSchema ::= SEQUENCE
{
  key_server_provider_identifier    UTF8String (SIZE (1..32))
  service_provider_identifier       UTF8String (SIZE (1..32))
  service_identifier                UTF8String (SIZE (1..32))
}
```

1

Listing 11-9: SBxD/KIS Endpoint Certificate Extension Data

```
1 sbxdkis-cert-extension-data SBxDKisCertificateExtensionSchema ::=
2 {
3   key_server_provider_identifier    '...', --value shall be the CCC-assigned 4-letter identifier
4   service_provider_identifier       '...' --identifies the type of service provider as a business entity
5   service_identifier               '...' --identifies the type of service provided by the service provider
6                                   --defined outside of CCC
7 }
```

2

3

Listing 11-10: SBxD/KIS Endpoint Certificate Data

```
1 sbxdkis-cert-data Certificate ::=
2 {
3   tbsCertificate
4   {
5     version v3, --shall be v3
6     serialNumber ..., --a random integer chosen by the certificate issuer
7     signature
8     {
9       algorithm {1 2 840 10045 4 3 2} --OID for ecdsaWithSHA256 (ANSI X9.62 ECDSA algorithm with SHA256)
10    },
11    issuer rdnSequence:
12    {
13      {
14        {
15          type {2 5 4 3}, --OID for CommonName
16          value "..." --shall match the subject of the issuer certificate
17          --(shall use UTF8String format)
18        }
19      }
20    },
21    validity
22    {
23      notBefore Time: "...", --shall use UTCTime or GeneralizedTime as defined in [3]
24      notAfter Time: "...", -- shall use UTCTime or GeneralizedTime as defined in [3]
25    },
26    subject rdnSequence:
27    {
28      {
29        {
30          type {2 5 4 3}, --OID for CommonName
31          value "..." --contains the subject of the certificate
32          --(shall use UTF8String format)
33        }
34      }
35    },
36    subjectPublicKeyInfo
37    {
38      algorithm
39      {
40        algorithm {1 2 840 10045 2 1}, --OID for ecPublicKey (ANSI X9.62 public key type)
41        parameters {1 2 840 10045 3 1 7} --OID for prime256v1(ANSI X9.62 named elliptic curve)
42      },
43      subjectPublicKey '04...'H
44      --the public key pre-pended with 04 to indicate uncompressed format
45    },
46    extensions
```



```

47 {
48 {
49   extnID    {install_param_oid_endpoint}, --OID for Endpoint certificate
50   critical  TRUE,
51   extnValue  '...'H --DER encoding for sbxdkis-cert-extension-data as per Listing 11-8
52 },
48 {
49   extnID    {1.3.6.1.4.1.41577.5.16}, --OID for SBxD-KIS certificate
50   critical  TRUE,
51   extnValue  '...'H --DER encoding for sbxdkis-cert-extension-data as per Listing 11-8
52 },
53 {
54   extnID    {2 5 29 15}, --KeyUsage standard extension
55   critical  TRUE,
56   extnValue  '03020204'H --DER encoding for KeyUsage, digitalSignature
57 },
58 {
59   extnID    {2 5 29 19}, --BasicConstraints standard extension
60   critical  TRUE,
61   extnValue  '30060101FF020100'H -- DER encoding for BasicConstraints CA=TRUE, pathLenConstraint=0
62 },
63 {
64   extnID    {2 5 29 35}, --OID for AuthorityKeyIdentifier standard extension
65   critical  FALSE,
66   extnValue  '...'H -- DER encoding of an AuthorityKeyIdentifier sequence as defined in \[3\], containing only
67               -- a KeyIdentifier element. The KeyIdentifier is an OCTET STRING containing the
68               -- 160-bit SHA-1 hash of the value of the BIT STRING subjectPublicKey
69               --from the issuer certificate (excluding the tag, length, and number of unused bits)
70 },
71 {
72   extnID    {2 5 29 14}, --OID for SubjectKeyIdentifier standard extension
73   critical  FALSE,
74   extnValue  '...'H --160-bit SHA-1 hash of the value of the BIT STRING subjectPublicKey
75               --(excluding the tag, length, and number of unused bits)
76 },
77 {
78   extnID    {...}, --Optional extensions added by the issuer, content not verified by the applet
79   critical  FALSE,
80   extnValue  '...'H --DER encoding of the extension
81 },
82 ...
83 }
84 },
85 signatureAlgorithm
86 {
87   algorithm {1 2 840 10045 4 3 2}
88 },
89 signatureValue '...'H --the certificate signature by the issuer computed as per RFC 5280 \[3\]
90 }

```

1 *Listing 11-11: SBxD/KIS Intermediate CA Certificate Extension Schema*

```

1 sbxdkis-intermediateCACertificateExtensionSchema ::= SEQUENCE
2 {
3   extension_version    INTEGER (1..255),
4 }

```

Listing 11-12: SBxD/KIS Intermediate CA Certificate Extension Data

```
1 sbxdkis-intermediate-ca-cert-extension-data IntermediateCACertificateExtensionSchema ::=
2 {
3   extension_version      1, --value shall be 1
4 }
```

Listing 11-13: SBxD/KIS Intermediate CA Certificate Data

```
1 sbxdkis-intermediate-ca-cert-data Certificate ::=
2 {
3   tbsCertificate
4   {
5     version v3, --shall be v3
6     serialNumber ..., --a random integer chosen by the certificate issuer
7     signature
8     {
9       algorithm {1 2 840 10045 4 3 2} --OID for ecdsaWithSHA256 (ANSI X9.62 ECDSA algorithm with SHA256)
10    },
11    issuer rdnSequence:
12    {
13      {
14        {
15          type {2 5 4 3}, --OID for CommonName
16          value "..." --shall match the subject of the issuer certificate
17            --(shall use PrintableString or UTF8String format)*
18        }
19      }
20    },
21    validity
22    {
23      notBefore Time: "...", --shall use UTCTime or GeneralizedTime as defined in [3]
24      notAfter Time: "...", -- shall use UTCTime or GeneralizedTime as defined in [3]
25    },
26    subject rdnSequence:
27    {
28      {
29        {
30          type {2 5 4 3}, --OID for CommonName
31          value "..." --contains the subject of the certificate (intermediate_CA_identifier)
32            --(shall use UTF8String format)
33        }
34      }
35    },
36    subjectPublicKeyInfo
37    {
38      algorithm
39      {
40        algorithm {1 2 840 10045 2 1}, --OID for ecPublicKey (ANSI X9.62 public key type)
41        parameters {1 2 840 10045 3 1 7} --OID for prime256v1(ANSI X9.62 named elliptic curve)
42      },
43      subjectPublicKey '04...'H
44      --the public key pre-pended with 04 to indicate uncompressed format
45    },
46    extensions
47    {
```

```

48 {
49     extnID    {1.3.6.1.4.1.41577.5.15}, --OID for Intermediate CA extension
50     critical  TRUE,
51     extnValue '...'H --DER encoding for intermediate-ca-cert-extension-data as per Listing 15-15
52 },
53 {
54     extnID    {2 5 29 15}, --KeyUsage standard extension
55     critical  TRUE,
56     extnValue '03020204'H --DER encoding for KeyUsage, CertSign only
57 },
58 {
59     extnID    {2 5 29 19}, --BasicConstraints standard extension
60     critical  TRUE,
61     extnValue '30060101FF020100'H -- DER encoding for BasicConstraints CA=TRUE, pathLenConstraint=0
62 },
63 {
64     extnID    {2 5 29 35}, --OID for AuthorityKeyIdentifier standard extension
65     critical  FALSE,
66     extnValue '...'H -- DER encoding of an AuthorityKeyIdentifier sequence as defined in \[3\], containing only
67         -- a KeyIdentifier element. The KeyIdentifier is an OCTET STRING containing the
68         -- 160-bit SHA-1 hash of the value of the BIT STRING subjectPublicKey
69         --from the issuer certificate (excluding the tag, length, and number of unused bits)
70 },
71 {
72     extnID    {2 5 29 14}, --OID for SubjectKeyIdentifier standard extension
73     critical  FALSE,
74     extnValue '...'H --160-bit SHA-1 hash of the value of the BIT STRING subjectPublicKey
75         --(excluding the tag, length, and number of unused bits)
76 },
77 {
78     extnID    {...}, --Optional extensions added by the issuer, content not verified by receiver
79     critical  FALSE,
80     extnValue '...'H --DER encoding of the extension
81 },
82 ...
83 }
84 },
85 signatureAlgorithm
86 {
87     algorithm {1 2 840 10045 4 3 2}
88 },
89 signatureValue '...'H --the certificate signature by the issuer computed as per RFC 5280 \[3\]
90 }

```

* The selection of format between PrintableString and UTF8String is outlined in Section [B.2.4](#) of Appendix B

1

2 11.13 Prerequisites

- 3 1. The Digital Key applet helper methods described in Section [15.4](#) or equivalent
- 4 implementation are available on receiver and sender device.
- 5 2. Sender and receiver devices contain the necessary key material to authenticate remote
- 6 servers. This key material is stored on the Digital Key framework.
- 7 3. Sender and receiver Digital Key applet or equivalent implementation possess an Instance
- 8 CA dedicated to the Vehicle OEM brand involved in sharing. The issuance of Instance CA
- 9 is described in Section [16](#).

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
4. Sender and receiver devices have retrieved an External CA certificate as defined in Section 16.2.6. This certificate is stored in the Digital Key framework of both devices.
5. The owner pairing has been successfully executed on the owner device, as per Section 6.
6. If applicable, immobilizer tokens and slot identifiers are available on the owner device’s confidential and private mailboxes, as per Section 6.
7. The following values have been obtained by the owner, as per Section 6 or equivalent preliminary operations. These values are stored on the Digital Key framework:
- (a) IMMOBILIZER_TOKEN_LENGTH
- (b) SIG_BMP_OFFSET
- (c) SLOT_ID_OFFSET
- (d) VEHICLE_OEM_PROPRIETARY_DATA_OFFSET
- (e) KEY_ATT_OFFSET
- (f) MAILBOX_CONTENT_END_OFFSET
- (g) LONG_TERM_SHARED_SECRET
- (h) ROUTING_INFORMATION
- (i) DIGITAL_KEY_APPLET_PROTOCOL_VERSIONS (V-D-TX-vehicleList)
- (j) SHARING_CONFIGURATION
8. Sender and receiver OEM Server have implemented the Key Life Cycle Management functions especially key termination and deletion as described in section 13.

20

11.14 Error Codes

21

22

23

24

25

The following codes may be sent from sender device to receiver device or vice versa if an error occurs during the sharing process. The sharing procedure shall be aborted if an error is returned by receiver or owner.

The error codes are returned in the TLV structure as shown in Table 11-26. All error codes that are unknown to the receiver shall be treated as FF_h (unspecified error).

26

Table 11-26: Error Code Message

Tag	Length (bytes)	Description	Field is	Domain Version
7F6C _h	variable	Error Message		OD-FD-KS
5A _h	1	Error code as per Table 11-27	mandatory	

27

28

29

30

If a key is shared with or from a device does not yet support error code Tag 5Ah, the key sharing error code messages shall contain the following byte string (which is a non-compliant TLV structure) instead:

- 31
- 7F6C036d01xxh, where xx is the error code as per Table 11-27.

32

Sender devices that do not support error code Tag 5Ah can be identified by:

- 33
- 34
- 35
- OD-FD-KS < 0300h in Sub tag 54h of Key Creation Request (7F31h), or
- not sending supported OD-FD-KS versions sub tag (54h) in Key Creation Request (7F31h)

36

Receiver devices that do not support error code Tag 5Ah can be identified by:

- 37
- OD-FD-KS < 0300h in Sub tag 55h of Key Signing Request (7F36h), or

- not sending supported OD-FD-KS versions sub tag (55h) in Key Signing Request (7F36h)

Note: If sharing needs to be cancelled and recipient has not yet sent the key signing request to the sender, the sender device cannot determine the OD-FD-KS domain version. Therefore, to cancel sharing the sender can include either Subtag 5A_h or 6D_h in the error message (Tag 7F6C_h).
Receivers with OD-FD-KS ≥ 0300_h shall be capable of receiving and parsing this message.

Table 11-27: Error Codes

Error code	Description
00 _h	Missing mandatory field
01 _h	Invalid message structure
02 _h	Invalid message content
03 _h	Version not supported
04 _h	Certificate expired
05 _h	Invalid certificate structure
06 _h	Invalid certificate content
07 _h	Invalid certificate chain
08 _h	Request rejected
09 _h -1F _h	RFU
20 _h	Sharing cancelled
21 _h	Sharing cancelled by owner – no valid Pin received after max number of attempts
22 _h	Sharing cancelled by friend
23 _h -FE _h	RFU
FF _h	Unspecified error

11.15 Algorithms

Listing 11-14: setPrivateMailboxBit

```

input: endpoint, bit_index, offset
output: n/a
begin
  execute endpoint.getPrivateData as per Section 15.4.1.17
    input: offset, length = 1
    output: one_byte

  mask = 1
  mask = mask << bit_index
  one_byte = one_byte | mask

  execute endpoint.setPrivateData as per Section 15.4.1.18
    input: offset, data = one_byte
end

```

1 *Listing 11-15: clearPrivateMailboxBit*

```
1 input: endpoint, bit_index, offset
2 output: n/a
3 begin
4   execute endpoint.getPrivateData as per Section 15.4.1.17
5     input: offset, length = 1
6     output: one_byte
7
8   mask = 1
9   mask = mask << bit_index
10  mask = ~mask
11  one_byte = one_byte & mask
12
13  execute endpoint.setPrivateData as per Section 15.4.1.18
14    input: offset, data = one_byte
15  end
```

2 *Listing 11-16: clearPrivateMailboxBytes*

```
1 input: endpoint, offset, length
2 output: n/a
3 begin
4   generate empty_buffer using length
5   execute endpoint.setPrivateData as per Section 15.4.1.18
6     input: offset, data = empty_buffer
7  end
```

3 *Listing 11-17:Generate Attestation Signature*

```
1 input: data_fields, private_key
2 output: signature
3 begin
4   set curve_parameters ← "ECC NIST P-256" as per \[8\]
5   signature ← ECDSA(curve_parameters, private_key, data_fields) using SHA-256 as per \[8\]
6   return signature (64 bytes)
7  end
```

4 11.16 Versioning

- 5 Key sharing data is exchanged via framework APIs and is mostly consumed by the applet. The
6 version information shall combine framework and applet version into OD-FD-KS.
- 7 Version agreement does not add another roundtrip to the sharing flow. Instead, the version
8 information is sent by the sender device as part of the key creation request. This implies that the
9 provided key creation data can be consumed by the receiver device independently of the
10 supported receiver versions. To achieve this, the key creation data must contain all data required
11 by all possibly supported receiver device versions.
- 12 If, for example, the format of the endpoint identifier changes in a future version, the new
13 endpoint identifier must be identified by a new tag and the owner device must send old and new
14 identifiers. The receiver device can then pick the required format.
- 15 Mandatory tags shall not be removed from the key creation request, as the receiver device might
16 only support a version where this tag is required to proceed.

- 1 The receiver device determines the highest commonly supported version from the owner version
- 2 list and its own version list. It then sends the key signing request in a format supported by the
- 3 owner device.
- 4 The receiver device uses the list of vehicle-supported owner pairing versions (V-OD-FW) to
- 5 determine the format of the key tracking request ([Table 11-17](#)). Note that content and format of
- 6 some fields of the key tracking request are determined by other domain versions.
- 7 If no common version can be determined, then both sides shall fall back to the sharing data
- 8 formats defined before versioning was introduced

12 SERVER-BASED KEY SHARING

This chapter defines how an SBOD or SBFD communicates with its connected actors. If the standardized API between Vehicle OEM Server and SBOD or SBFD as defined in this specification is replaced by proprietary APIs, or any standardized API of SBOD or SBFD is amended by proprietary APIs, then such proprietary APIs may be used (i.e., implementations will not be deemed non-compliant solely due to use of such proprietary APIs).

12.1 Server Preparation

The process of establishment of a connection between the FMS/SPS and the SBOD/SBFD (link 21, link 22), as well as any certificate exchange, allow-listing, etc. is out of scope of this specification. Since Digital Keys are transferred over links 21 and 22 ([Figure 2-1](#)), to ensure security, these links shall be protected using 2-way TLS protocol. Once established, digital key specific communication between server entities should be conducted using APIs described in [Section 17](#).

12.1.1 FMS/SBOD Connection Preparation (example)

During the preparation phase, the FMS and the SBOD set-up a secure server-to-server connection, e.g., by using 2-way TLS.

1. The FMS requests a connection to the SBOD via Transport Control Protocol (TCP).
2. The TCP connection request is accepted by the SBOD and the connection is opened.
3. The SBOD requests a certificate and key exchange with the FMS.
4. To sign the certificate a public key is published to the FMS.
5. The FMS key is verified and signed with the server public key.
6. The FMS publishes the signed key to the SBOD.
7. The signature is verified and the FMS key accepted.
8. An encrypted connection is established between the FMS and the SBOD.

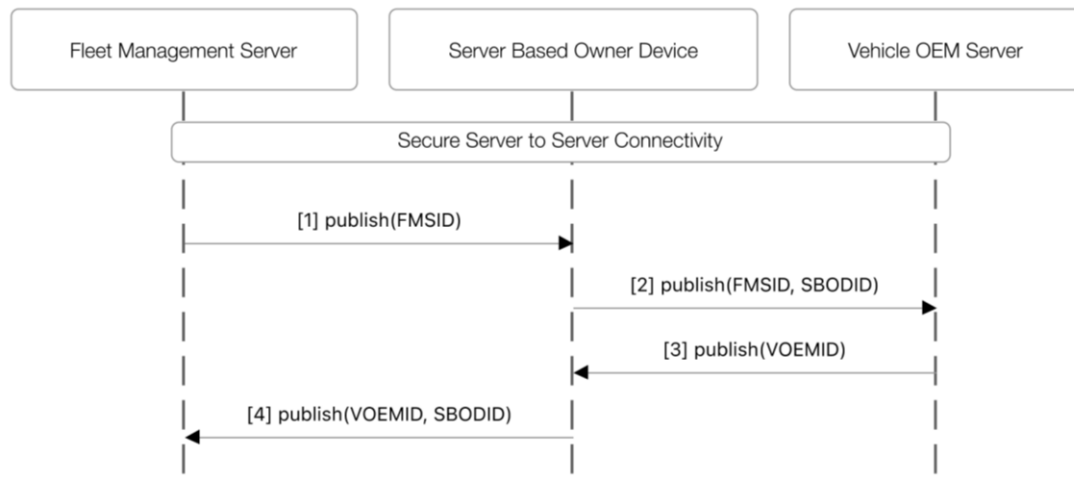
After the establishment of the link a secure data exchange to the trusted FMS is possible.

12.1.2 FMS/SBOD/Vehicle OEM Server Preparation (example)

During the preparation phase, it is assumed that the FMS, SBOD and the vehicle OEM server have an established secure server to server connectivity. To cumulate all necessary information between the FMS, the SBOD and the vehicle OEM server, the following data for identification may be exchanged using links 21 and 20 of [Figure 2-1](#), as defined in the high-level architecture:

1. FMS unique identifier, allocated by vehicle OEM.
2. SBOD provides the FMS unique identifier, allocated with the identifier of the SBOD.
3. The vehicle OEM server provides its unique ID to the SBOD.
4. Unique Identifier of the SBOD, allocated with the vehicle OEM ID.

Figure 12-1: Server Preparation



12.2 Vehicle Infleeting

12.2.1 Proof of Ownership

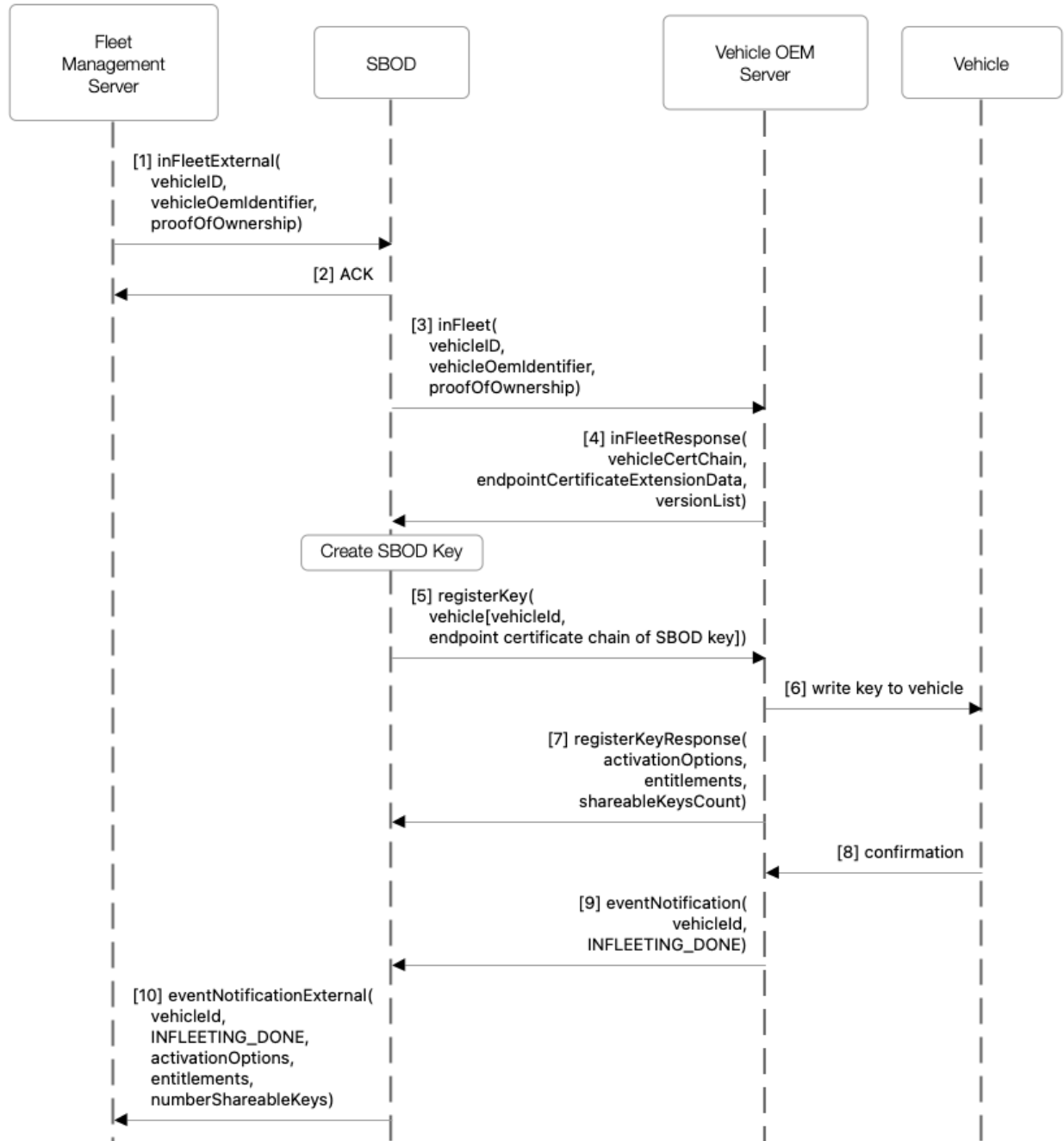
As part of the infleeting process, to create an association between the FMS and a vehicle, verification of proof of ownership shall be performed. The exact mechanism for verification of proof of ownership by the FMS/SPS may or may not involve the Vehicle OEM directly and is out of scope of this specification. The vehicle OEM shall have the ability to determine the acceptable methods to prove ownership. During the infleeting process, if requested by the Vehicle OEM server, the SBOD shall collect and verify the proof of ownership and provide it to the Vehicle OEM.

12.2.2 Infleeting Flow

Figure 12-2 shows the infleeting process when started by the FMS

1

Figure 12-2: Pairing Flow for SBOD



2

3 12.2.2.1 Steps 1 and 2: FMS Infleeting Request

4 The FMS requests the SBOD to start the 3rd Party infleeting process for the vehicle using the
5 inFleetExternal() API. The request contains vehicleOemIdentifier, vehicleIdentifier and
6 optionally proof of ownership. The SBOD acknowledges the start of the infleeting process.

12.2.2.2 Steps 3 to 9: SBOD Key Deployment

The SBOD initiates the infleeting process with the Vehicle OEM Server by calling the `inFleet()` API including the VIN of the vehicle and Vehicle OEM Identifier and if required by Vehicle OEM policy, Proof of Ownership. The Vehicle OEM Server replies with `InFleetResponse()` to provide the following information to the SBOD:

1. Vehicle Certificate Chain
2. Endpoint certificate extension data and
3. The list of the supported versions of the vehicle in the `versionList`.

After creating the endpoint, the SBOD shall send the `registerKey()` request to the Vehicle OEM Server. The request contains the endpoint certificate chain of the SBOD key. The vehicle OEM server shall deploy the SBOD key to the vehicle and send a `registerKey()` Response message to the SBOD. When writing the key to the vehicle, the vehicle OEM server also provides Device Configuration Data (Table 5-14) to vehicle. The `registerKey()` response contains vehicle fleet capabilities (including `activationOptions`, `entitlements` and `numberShareableKeys`). This information is needed for the SBOD to generate a `KeyCreationRequest`, thus allowing creation of shared keys as needed up to `numberShareableKeys`. After the vehicle confirms the SBOD key, the vehicle OEM server sends an `eventNotification()` Request with status `INFLEETING_DONE` to the SBOD.

If the SBOD is implemented by the Vehicle OEM, then SBOD Key Deployment: Steps 3 to 9 may be done using proprietary mechanism.

12.2.2.3 Step 10: Infleeting Confirmation

The SBOD notifies FMS about the successful infleeting process and forwards vehicle capabilities for fleet received in `registerKey()` Response to FMS.

12.3 Digital Key Sharing for Fleet Vehicles

12.3.1 Receiver device Binding

To achieve a high level of security, an account binding between FMS user ID and Instance CA of the user's device may be performed. This process is identical to the vehicle attestation process described in Section titled "Vehicle Attestation" of [41]. After successfully sharing a key as defined in Chapter 11, the SBOD can store the public key of the Instance CA for that user after receiving the vehicle attestation. In subsequent sharing sessions, the SBOD may verify the same Instance CA public key in the `keySigningRequest`. (Note: Instance CA may be deleted and recreated between two sharings if the deletion results in the deletion of the last key for that vehicle brand on the device).

12.3.2 Receiver device Sharing App Access Rights

Apps which have been agreed upon (The agreement process is outside the scope of this specification) that redeem the sharingURL on the receiver device, shall have implicit access rights to the Digital Key.

Note, that the Vehicle OEM application shall always have implicit access rights to any Digital Key associated with the corresponding vehicle OEM.

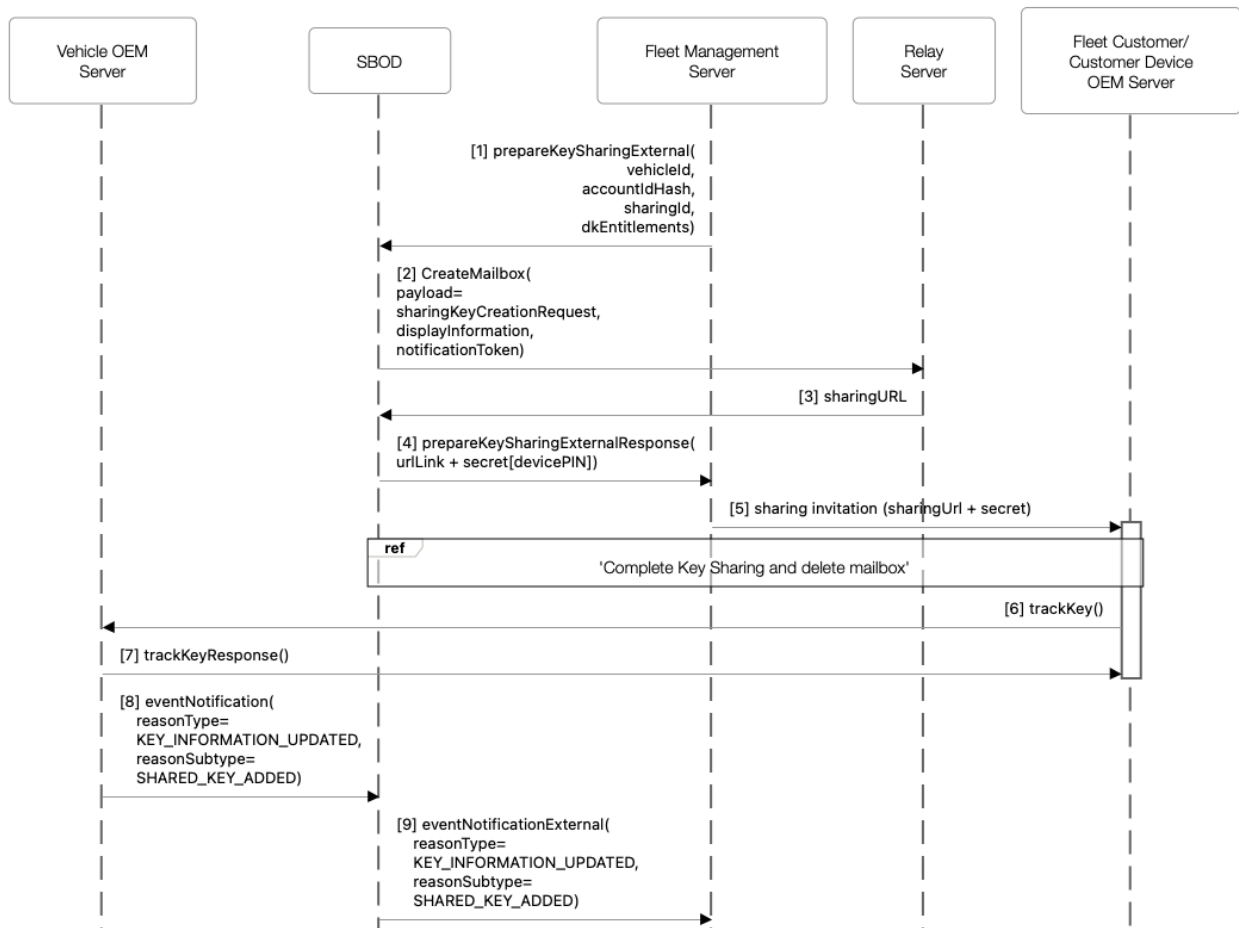
12.3.3 Digital Key Sharing Flow

When a vehicle is allocated to a user, the FMS initiates sharing of a receiver Digital Key from the SBOD to the user device. The process is analogous to the cross platform sharing process described in Chapter 11, where a relay server is used to share a Digital Key.

The fleet vehicle shall not allow the receiver device to unlock, start, or perform other vehicle functions until a valid key tracking receipt has been obtained from the KTS if Tag DB_h is included in the device configuration data (Table 5-14) provided by the vehicle OEM server during infleeting.

The Digital Key sharing flow is described in detail in the following sequence diagram:

Figure 12-3: Digital Key Sharing Flow for Fleet Vehicles



For a vehicle, several Digital Keys may be shared with different user devices (FMS staff, fleet vehicle user, and fleet vehicle user's friend). The flow described in [Figure 12-3](#) is executed for each device.

12.3.3.1 Steps 1 through 4: Digital Key Sharing URL Request

When the FMS needs to share a Digital Key to a vehicle user, FMS shall request the SBOD to generate a sharingURL and provide the SBOD with sharingId, vehicleId, selected entitlements and the AccountInfoHash of the user device which will receive the friend DK.

The SBOD creates a mailbox on an approved relay server as described in Section [11.4](#) and obtains a sharingURL from the relay server. Upon reception of the sharingURL from the Relay Server, the SBOD shall send a prepareKeySharingExternal() Response message to the FMS including the sharingURL, secret and if required by Vehicle OEM key activation mechanisms such as sharingPassword or devicePIN.

12.3.3.2 Step 5: Channel Establishment

The FMS generates a sharing invitation and forwards it to the user Device OEM server over a proprietary interface. On reception of the sharingURL the receiver device executes cross platform key sharing as described in Chapter 11 with the SBOD serving the role of owner device.

12.3.3.3 Steps 6 through 7: Key Tracking

After key sharing as per Chapter 11 is finished, if required by the Vehicle OEM, to complete registration of the receiver Digital Key with the vehicle OEM KTS, the receiver device OEM server shall send a trackKey request to the vehicle OEM server.

12.3.3.4 Steps 8 through 9: FMS Notification

The Vehicle OEM Server shall inform the SBOD that a new shared key was added using an eventNotification() Request. The SBOD may inform the FMS that a new shared key was added using eventNotificationExternal or other proprietary mechanisms.

If required by the vehicle OEM or fleet management, the vehicle OEM server may also send additional proprietary event notifications to inform the FMS of the new sharing request.

12.4 Delegated Key Sharing

12.4.1 SBFD Enablement

To enable a Service Provider (e.g., 3rd Party peer-to-peer car sharing, vehicle OEM garage or road assistance) to issue keys for certain vehicles to its subscribers or customers, the service provider shall be connected to a Server Based Friend Device (SBFD). Business relationships between the Service Provider and the SBFD hosting entity, as well as between the SBFD hosting entity and the Vehicle OEM need to be established. The process of establishing these relationships are out of scope of this specification.

Each SBFD needs to be enabled by the Vehicle OEM. To do so, the root CA Certificate of the SBFD and the root CA certificate of the Vehicle OEM Server need to be cross signed to obtain the external CA certificate. The external CA Certificate is used by the SBFD to create an (optional) intermediate CA certificate, which then creates an SBxD Key Server Certificate, whose private key will be used to sign the shared keys. The SBxD Key Server Certificate contains SBXD ID, Service Provider ID and Service ID (e.g. car wash service). The Service Provider shall be identified by a Service Provider ID and shall ensure that unique Service IDs are used for each service offered.

Upon enablement of a service, a holder of a digital key with appropriate sharing rights (entitlements) and sharing activated for vehicle (sharing control allows accepting shared keys) can choose to allow a specific Service Provider to request keys for a specific service for the vehicle associated with that digital key. This can be done by selecting the Service Provider from a list provided by the Vehicle OEM in an App or web interface.

12.4.2 SBFD service activation

Once SBFD enablement has completed, the SBFD service can be activated by the owner or authorized users. The activation can be accomplished by sharing a key with the corresponding account role setting as outlined in Section [2.8.4](#).

In addition to the standard validation and checks of receiver keys as defined in Appendix A.1.81.a.i.A.4.7, during tracking of the SBxD.PK, the KTS also verifies that the certificate chain communicated in [Table 11-5](#) is valid.

If the SBFD is implemented by the Vehicle OEM, service activation can be done using one of the following methods:

1. By the owner or authorized users through sending a standardized service management request (see [Table 12-1](#)) to the SBFD. This method is described in section [12.4.2.1](#).
2. By proprietary methods which may include but are not limited to: Web-Portal login, verification of other cryptographic signatures or material, vehicle registration document verification, etc. The definition of these methods is out of scope of this specification.

12.4.2.1 Standardized SBFD Service Activation

This method includes using the Vehicle OEM app on a device that holds a Digital Key to the vehicle for which the service shall be activated.

The Vehicle OEM App shall provide Tag 7F63_h (secure sign data) to the Secure Element on the device to create Tag 7F62_h (Service Management Request Data) out of it. Tag 7F63_h (secure sign data) defines two optional parameters to append additional data to be included into the Service Management Request (7F64_h): Tag BF24_h (Server-specific parameters) can be directly provided by the Vehicle OEM server while compiling tag Tag 7F63_h (secure sign data) and Tag BF25_h (Client-specific parameters) can be used by the Vehicle OEM App to add additional data. The data is forwarded to the SE and the secure SIGN command is called. The SE of the device on which the App is installed creates the signature using the private key of the Digital Key associated with the vehicle. The Vehicle OEM server uses the signed Service Management Request Data to securely identify which service is to be enabled. The app receives the

serviceManagementRequest in the secure SIGN response, which will be forwarded by the Vehicle OEM app to the vehicle OEM server **for verification**.

Table 12-1: serviceManagementRequest

Tag	Subtag	Length (bytes)	Description	Field is	Domain Version
7F64 _h		Variable	Service Management Request	mandatory	D-VS
Service Management Request Data (Content of Tag 7F62 _h in Table 12-2)				mandatory	D-VS
	9E _h	64	signature over fields from Table 12-2 with the endpoint private key	mandatory	D-VS

Table 12-2: Service Management Request Data

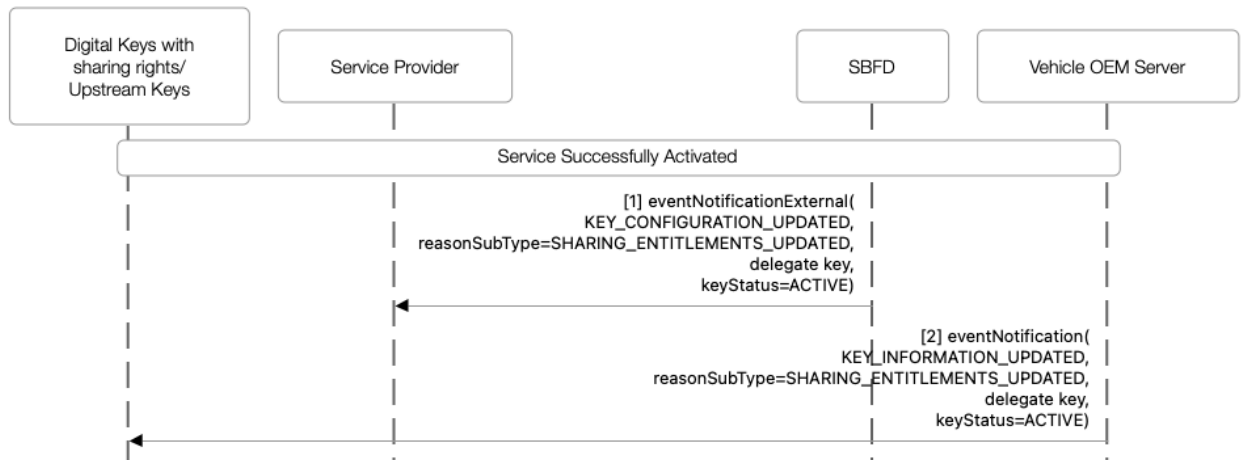
Tag	Subtag	Subtag	Length (bytes)	Description	Field is	Domain Version
7F62 _h			89	Service Management Request Data	mandatory	D-VS
	41 _h		1	Version = 01 _h version of the Service Management Request Data Fields	mandatory	
	92 _h		8	Random	mandatory	
	5D _h		20	key_identifier, 160-bit SHA-1 hash of the value of the BIT STRING subjectPublicKey from the endpoint certificate that issued the attestation (excluding the tag, length, and number of unused bits)	mandatory	
	7F63 _h		var	Secure Sign Data	mandatory	D-VS
		9F21 _h	16	ServiceProviderId	mandatory	D-VS
		9F22 _h	16	ServiceId	optional	D-VS
		9F23 _h	32	UsecaseToken	mandatory	D-VS
		BF24 _h	variable	Server-specific parameters, i.e. parameters set by the server to be displayed by the app to the user.	optional	D-VS
		BF25 _h	variable	Client-specific parameters, i.e. parameters set by the user via input form or otherwise. Appended by the app prior signing operation.	optional	D-VS
	93 _h		4	usage = D074DA4F _h , if user authentication was performed usage = FC6F4C17 _h , if user authentication was not performed	mandatory	D-VS

After positive verification of the serviceManagementRequest by the Vehicle OEM Server, the SBFD is enabled to issue keys for the vehicle.

12.4.2.2 Notifications

After successful service activation, the SBFD notifies the Service Provider using eventNotificationExternal API (Section 17.10.4.2). The Vehicle OEM server additionally notifies all upstream keys about the SBFD key sharing, using eventNotification API (Section 17.9.1.2).

Figure 12-4: eventNotification after Service Activation

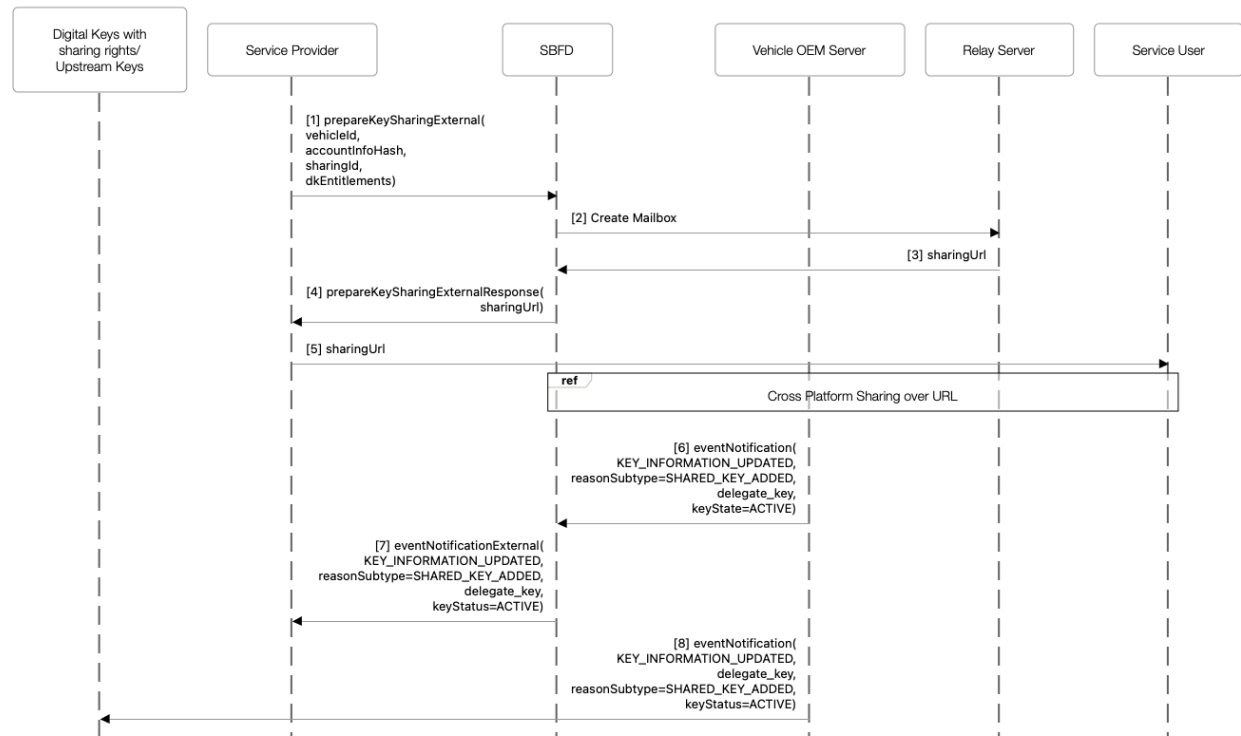


12.4.3 Service Key issuance from SBFD to Service User

The key sharing process for SBFD server follows the cross-platform key sharing as described in Chapter 11. Section 12.4.6 describes the considerations of Sharing Control by the owner.

To avoid performing a 2nd factor authentication, the Service Provider may deliver the AccountInfoHash of the receiver of the Digital Key to the SBFD. The decision to not require a second factor authentication based on AccountInfoHash is specific to the vehicle OEM.

Figure 12-5: Service Key sharing to Service User



12.4.3.1 Steps 1 to 4: Digital Key Sharing URL Request:

When the Service Provider needs to share a Digital Key to a service user, the Service Provider shall request the sharingURL from the SBFD using a prepareKeySharingExternal() Request and provides the sharingId, vehicleId and the (optionally) the AccountInfoHash of the user device which will receive the Digital Key. The SBFD creates a mailbox on an approved Relay Server as per Chapter 11. Once the SBFD receives a sharingUrl from the Relay Server it generates a prepareKeySharingExternal () Response with the sharingUrl and sends it to the Service Provider.

12.4.3.2 Steps 5: Digital Key Sharing:

The Service Provider sends the sharingUrl to the intended receiver device via a proprietary interface. On reception of the sharingURL the receiver device executes cross-platform key sharing as per Chapter 11 with the SBFD serving the role of the sender device and using the private key of its endpoint to sign the newly created shared key.

The receiver device completes the registration of the DK in the KTS of the vehicle OEM.

12.4.3.3 Steps 6 to 8: Notifications

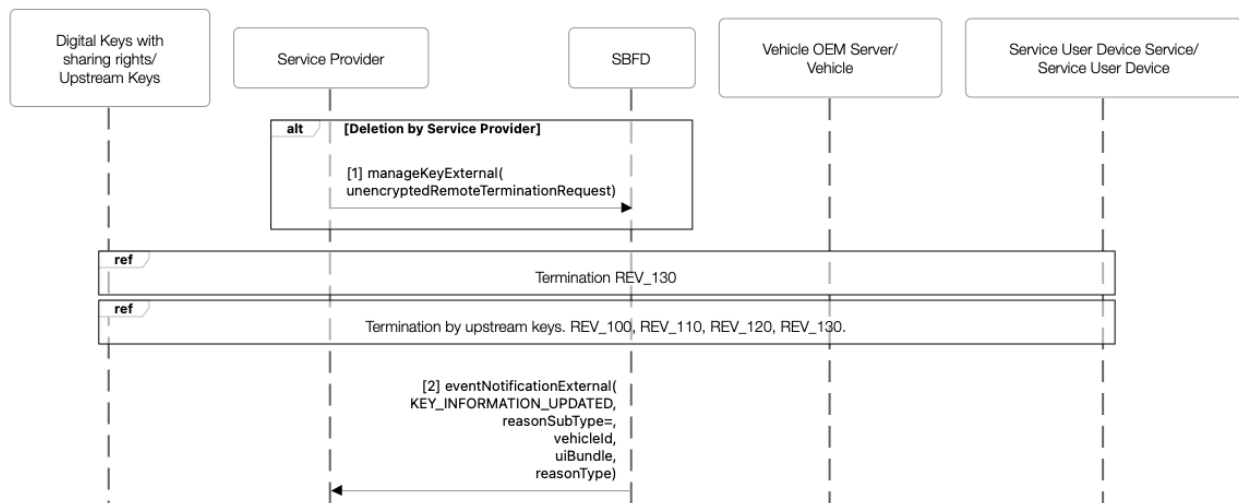
The Vehicle OEM Server informs the SBFD that a new shared key was added using eventNotification API (Section 17.9.1.2). If the SBFD is implemented by the Vehicle OEM the notification can be provided over a proprietary interface.

SBFD notifies the Service Provider about the successful sharing, using eventNotificationExternal API (Section 17.10.4.2). The Vehicle OEM server additionally notifies all upstream keys about the SBFD key sharing, using eventNotification API (Section 17.9.1.2).

12.4.4 Deletion of keys shared by SBFD

Deletion of keys shared by the SBFD can either be initiated either upon request of the Service Provider by calling the manageKeyExternal API, or by any eligible upstream key, Vehicle or Vehicle OEM Server.

Figure 12-6: Deletion of keys shared by SBFD



12.4.4.1 Step 1: Deletion Request

Deletion of keys shared by the SBFD can be triggered by the Service Provider using the manageKeyExternal API (Section [17.10.3.2](#)). The key deletion follows the Receiver Key Remote Termination of REV_130 with SBFD acting as Owner Device Server. The shared key can also be deleted by any upstream key, the vehicle or the vehicle OEM server (as described in Section [13.2](#) Remote Termination).

12.4.4.2 Step 2: Notification

After deletion of the key, SBFD notifies the Service Provider. (See [17.10.4](#))

12.4.5 Service Cancellation/SBFD Key Deletion:

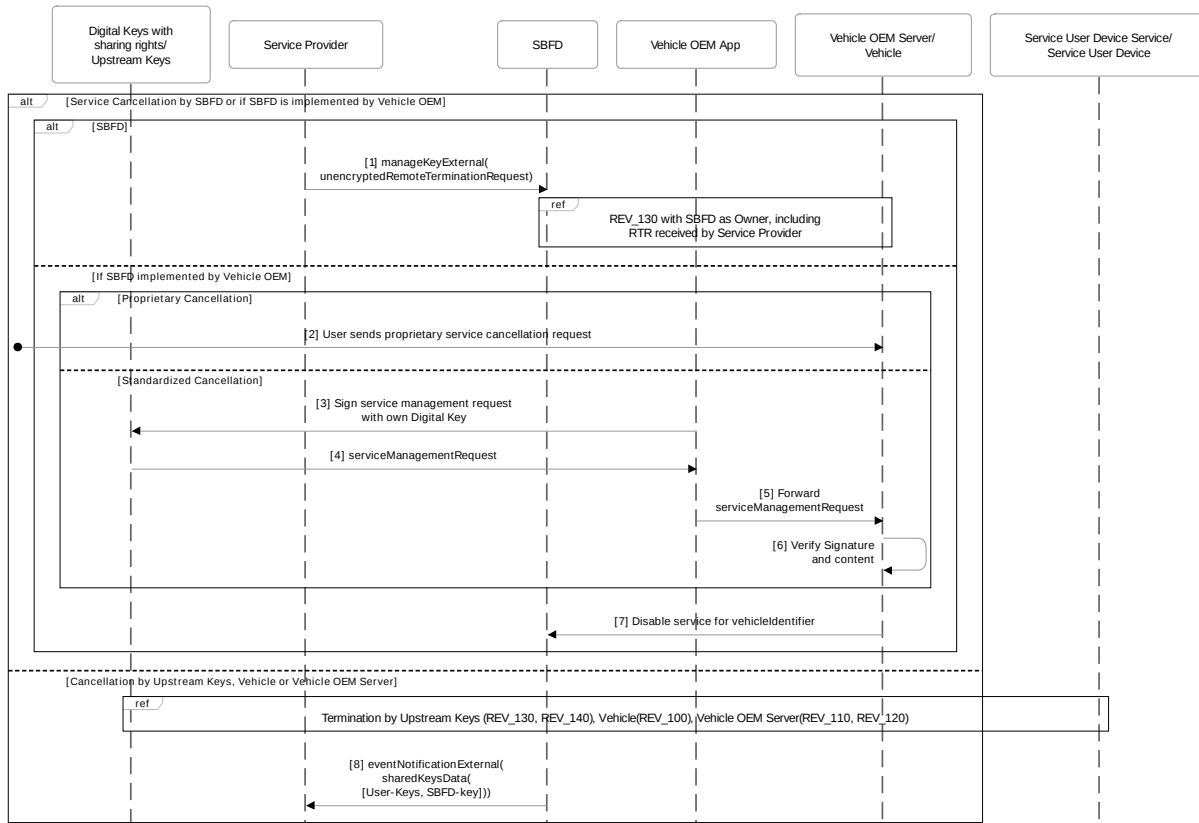
SBFD signing key is generated by the SBFD and transmitted to the vehicle when a user subscribes for a third party service. When a third-party service is cancelled, all keys associated with that SBFD shall be deleted before the SBFD signing key is deleted in the vehicle. This allows keys issued by the SBFD to devices to continue to be usable on the vehicle. If an SBFD is implemented by the Vehicle OEM, the service may be cancelled via the vehicle OEM app or proprietary means.

12.4.5.1 Step 1: Termination Request

Standardized Service cancellation shall be triggered by the Service Provider using the manageKeyExternal API (Chapter 17.7.9) which includes the Remote Server Key Termination Request that contains all keys shared from the SBFD, as well as SBFD's own server key. When the manageKeyExternal call arrives at SBFD, SBFD in turn shall create a single device Remote Termination Request for all keys contained in the received Remote Server Key Termination Request and especially its own SBFD owner key.

1

Figure 12-7: SBFD Service Cancellation



2

12.4.5.2 Steps 2 to 7:

If the SBFD is implemented by the Vehicle OEM, the service cancellation can be done by proprietary methods which may include but are not limited to: Web-Portal login, other cryptographic signatures or material, vehicle registration document verification, etc. Alternatively, standardized SBFD service cancellation can be performed by Upstream keys (13.2.4 REV_130, 13.2.5 REV_140), Vehicle (13.2.1 REV_100), or Vehicle OEM Server (13.2.2 REV_110, 13.2.3 REV_120).

12.4.5.3 Step 8: Service Provider Notification

After deletion of all keys, SBFD informs the Service Provider by calling the eventNotificationExternal API.

12.4.6 Owner Sharing Control

For owner-paired vehicles the owner and optionally Type 2 and Type 3 Digital Keys (see Table 2-3) or via vehicle OEM proprietary methods with sufficient proof of ownership should be provided with the option to disable and enable Digital Key sharing for their vehicle. This setting should be possible in the vehicle UI and from a device with owner, Type 2 and Type 3 Digital Keys. The vehicle OEM server can determine in the trackKey response whether a device is eligible to disable and enable Digital Key sharing. Once disabled, it shall not be possible to enable Digital Key sharing from a server, except from the vehicle OEM server under

specific conditions. The vehicle shall enforce the state where Digital Key sharing is disabled. It shall not be possible to enable Digital Key sharing through the vehicle OEM account, or the vehicle OEM app.

The vehicle should communicate the disabling of Digital Key sharing to the vehicle OEM server, which in turn should send an event notification to each device that has a Digital Key with sharing rights for the vehicle to indicate that Digital Key sharing is not possible. Devices should be notified in the same way if Digital Key sharing is enabled afterwards.

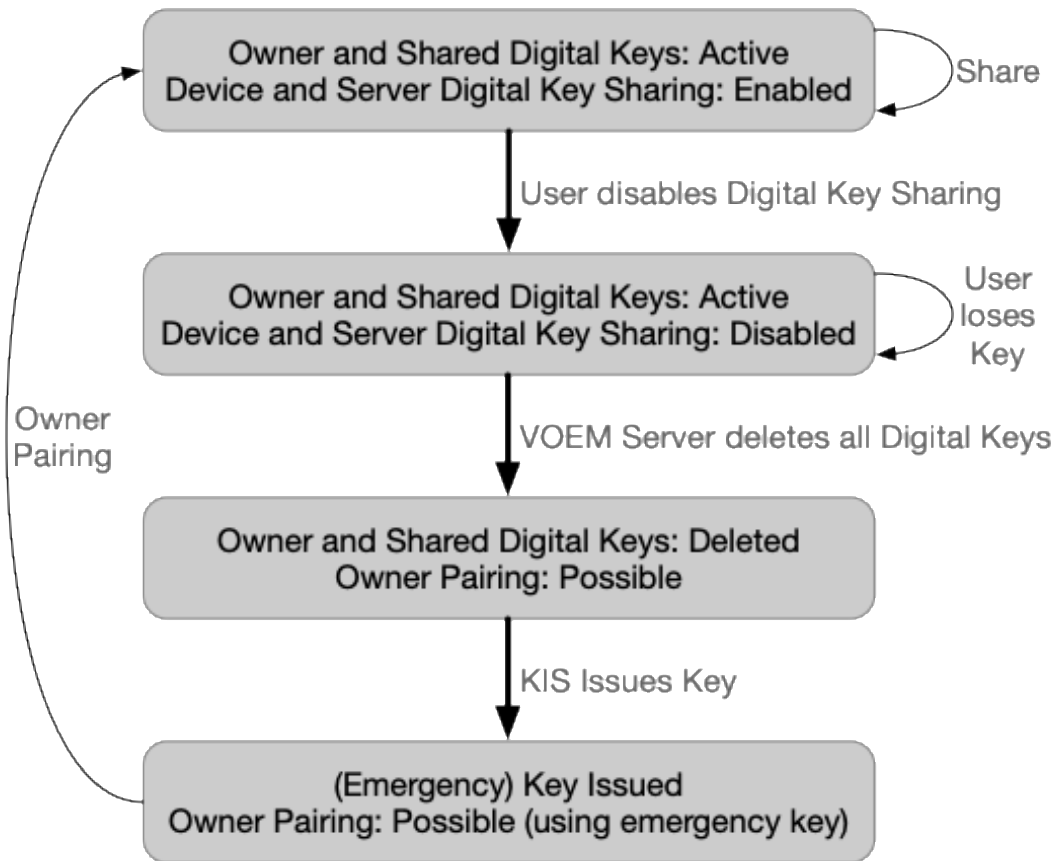
All shared Digital Keys that are known to the vehicle shall continue to work normally when Digital Key sharing is disabled. If a Digital Key is shared and tracked but not used on the vehicle before Digital Key sharing is disabled, then the vehicle should send a CONTROL FLOW P1 = 00_h, and P2 = A8_h (Digital Key sharing disabled, cannot accept the Digital Key right now). The device should show an appropriate notification to the user.

The factory reset by the Vehicle OEM server depicted in Figure 12-8 brings the vehicle into unpaired state where the KIS can issue and (emergency) key, for example to a plastic card. The factory reset shall be a distinct procedure to be executed by the KIS to enable Digital Key sharing. The vehicle shall not accept Digital Keys issued by the KIS while Digital Key sharing is disabled.

The vehicle OEM server shall inform all devices with Digital Key sharing rights and delegate servers via event notification “SHARING_POLICY_UPDATED” regarding the disable and enable status of Digital Key sharing. If a device has not (yet) received the event notification and tries to share the Digital Key by calling preShare(), the vehicle OEM server shall respond with 50132 (Digital Key Sharing Currently Disabled) sub-status error code (see [Table 17-68](#)).

1

Figure 12-8: KIS Factory Reset

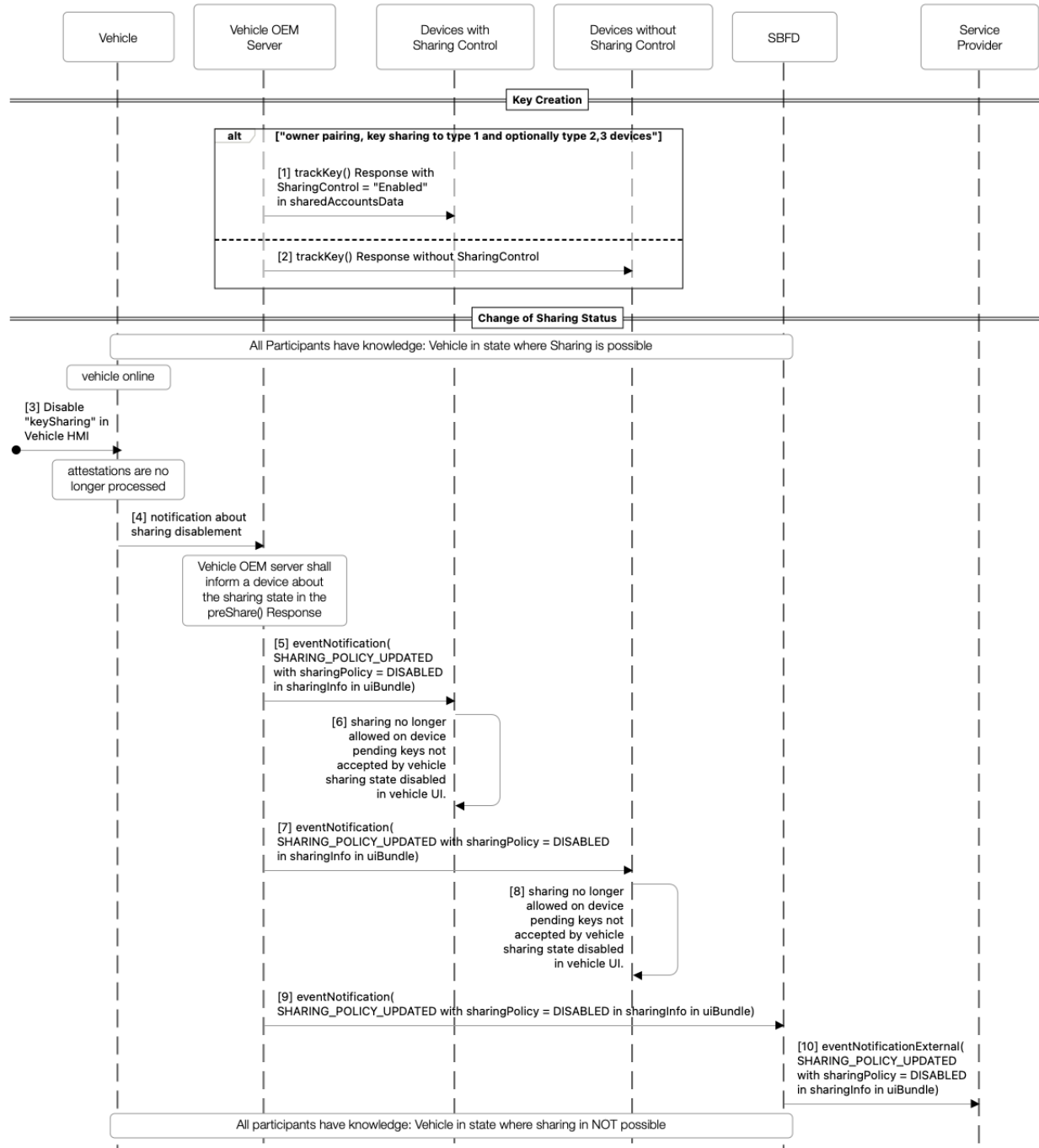


2

3

1

Figure 12-9: Sharing Control in Vehicle HMI



2

3

4

5

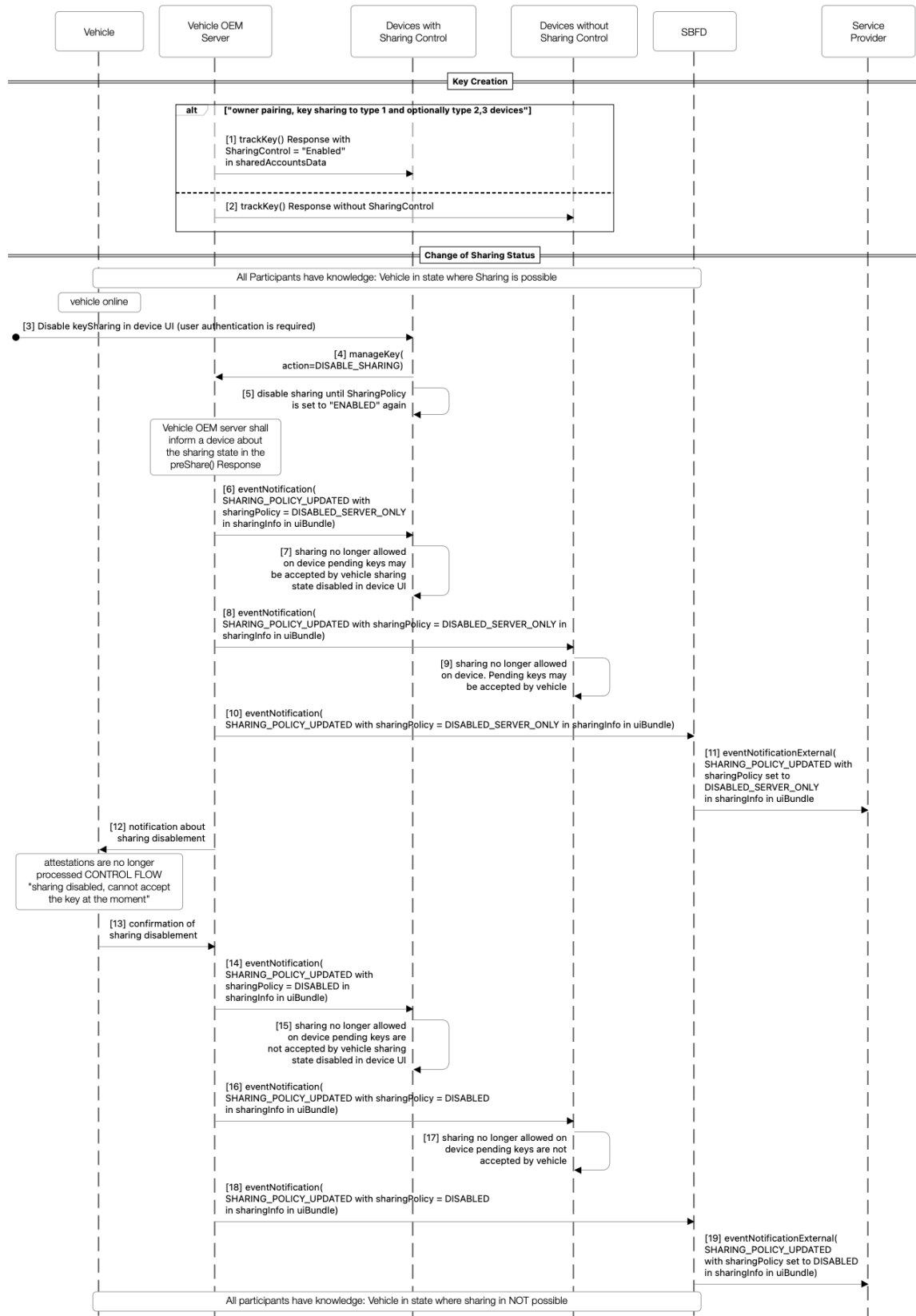
6

7

8

1

Figure 12-10: Sharing Control in Device UI



2

13 KEY TERMINATION AND DELETION [WCC1/WCC2]

Key termination may be triggered:

- in the vehicle, in the Vehicle OEM Server (e.g., account),
- on the device, and in the Device OEM Server (e.g., remote device wipe, if supported),
- using the service provider app or service provider server

The Vehicle OEM shall provide a mechanism for the user to terminate keys (owner and friend) by at least one of the following methods:

- Via an in-Vehicle user interface
- Via the Vehicle OEM Server (account)

In addition, the Vehicle OEM may provide a mechanism for the user to terminate keys (owner and friend) via

- the Vehicle OEM App , OR
- by other mechanisms. Which are out of scope of this specification

In addition, the Vehicle OEM shall provide a mechanism for the user to unpair using one or more of the following:

- via the Vehicle OEM App, or
- Via an in-Vehicle user interface, or
- Via the Vehicle OEM Server, or
- via other mechanisms which are out of scope of this specification.

The Device OEM shall provide a mechanism for the user to terminate keys (owner and friend) by at least one of the following methods:

- Via the Digital Key Native app, or
- Via the Device OEM Server (account), or
- Via other mechanisms which are out of scope of this specification.

A termination attestation should be sent from the device to the KTS whenever the device terminates the Digital Key. Along with the termination attestation, the device should send the Vehicle OEM proprietary data subsection of the private mailbox. The termination attestation and the Vehicle OEM proprietary data may not be sent in all cases due to technical constraints.

If the termination attestation is provided in the manageKey call from the Device OEM server to the Vehicle OEM Server, then the vehicle OEM server should consider the key as “terminated in the device” before attempting to terminate the key in the vehicle.

If the termination attestation is not provided in the manageKey call from the Device OEM server to the Vehicle OEM Server, then the vehicle OEM server shall continue to terminate the key in the vehicle while respecting the vehicle key termination conditions.

The Device OEM shall not terminate an active Digital Key unless one of the following conditions are met:

- The terminate action is initiated by an authenticated user and user intent on the device is confirmed, or

• After the vehicle OEM has signaled to the device OEM that the Digital Key has been deleted in the vehicle

In the case of remote device wipe (REV_170/REV_420), the action shall only be initiated by an authenticated user and Device OEM shall inform user that any keys on the device will not be useable.

To address user safety, a fade-out period may be established by the Vehicle OEM server. The fade out period is defined as the time between the IN_TERMINATION message and the actual termination of the key on the vehicle. The Vehicle OEM server notifies the sender or receiver device OEM server about the start of this period by sending an IN_TERMINATION notification. Further requirements associated with the fade-out period (e.g. use, purpose and duration of the period) are Vehicle OEM specific. A sample implementation of IN_TERMINATION notification is shown in [Table 17-56](#). After receiving the IN_TERMINATION message, Device OEM server should notify the user. Further behaviors after reception of the IN_TERMINATION message by the Device OEM server are specific to the Device OEM.

If a fade-out period is required, the Vehicle OEM should notify the device about the start of this period.

If the fade-out period has ended and/or the conditions for a successful termination of the key in the vehicle are met, the key is tracked as terminated in the KTS.

If online, sender, receiver, and all devices with visibility of the terminated keys shall be notified about the successful termination. The device should delete the associated Digital Keys prior to deletion of the instance CA triggered by DELETE CA command.

The detailed termination and deletion flows in the vehicle and device are described in Sections [13.2](#), [13.3](#), [13.4](#), [13.5](#), and [13.6](#).

13.1 Key Termination Scenarios

The termination and suspension use cases defined by CCC are listed in [Table 13-1](#), [Table 13-2](#), [Table 13-3](#), and [Table 13-4](#).

Remote termination scenarios (at minimum R-VT-2, R-DT-2, R-DT-4, R-VOT-9, R-VOT-10) should terminate keys for the vehicle on all devices belonging to the same device OEM account in a bulk (see section [14.2](#)).

Table 13-1: Vehicle-triggered Key Termination

Use Case ID	Trigger	Owner Key Status	Shared Key Status
R-VT-1	Owner terminates owner key in vehicle	Terminated	-
R-VT-2	Owner terminates shared key in vehicle	-	Terminated
R-VT-3	Owner changes owner device via vehicle	Terminated	-
R-VT-4	Owner/user unpairs vehicle from owner account or resets Digital Key function	Terminated	Terminated

Table 13-2: Device-triggered Key Termination

Use Case ID	Trigger	Owner Key Status	Shared Key Status
R-DT-1	Owner terminates owner key in Vehicle OEM app	Terminated	-

Use Case ID	Trigger	Owner Key Status	Shared Key Status
R-DT-2	Owner terminates shared key in Vehicle OEM app	-	Terminated
R-DT-3	Owner terminates owner key in native app	Terminated	-
R-DT-4	Owner terminates shared key in native app	-	Terminated
R-DT-5	Receiver terminates shared key in Vehicle OEM app	-	Terminated
R-DT-6	Receiver terminates shared key in native app	-	Terminated
R-DT-7	Owner unbinds vehicle from owner account in Vehicle OEM app	Terminated	Terminated

1

Table 13-3: Vehicle OEM-triggered Key Termination

Use Case ID	Trigger	Owner Key Status	Shared Key Status
R-VOT-1	Owner terminates owner key in Vehicle OEM customer web portal, hotline, or support	Terminated	-
R-VOT-2	Owner terminates shared key in Vehicle OEM customer web portal, hotline, or support	n/a	Terminated
R-VOT-3	Receiver terminates shared key in Vehicle OEM customer web portal, hotline, or technical support	n/a	Terminated
R-VOT-4	Owner Digital Key license expires	Terminated	Terminated
R-VOT-5	Terminate shared key after expiry date (automatic)	n/a	Terminated
R-VOT-6	Owner/legal authorities report a stolen/destroyed vehicle	Terminated	Terminated
R-VOT-7	Garage service process	Terminated	Terminated
R-VOT-8	Security breach on vehicle is detected.	Terminated	Terminated
R-VOT-9	Owner triggers deletion of personal data (owner account) in Vehicle OEM Server	Terminated	Terminated
R-VOT-10	Receiver triggers deletion of personal data (receiver account) in Vehicle OEM Server	n/a	Terminated
R-VOT-11	Owner unbinds vehicle from owner account in Vehicle OEM Server	Terminated	Terminated

2

Table 13-4: Device OEM-triggered Key Termination

Use Case ID	Trigger	Owner Key Status	Shared Key Status
R-DOT-1	Owner (sender)/User wipes device (factory reset)	Terminated	-
R-DOT-2	Receiver wipes receiver device (factory reset)	n/a	Terminated
R-DOT-3a	Sender reports lost/stolen/rediscovered owner device: suspend/resume keys	Suspended/ Resumed	n/a
R-DOT-3b	Sender reports lost/stolen owner device: remote wipe keys	Terminated	n/a

Use Case ID	Trigger	Owner Key Status	Shared Key Status
R-DOT-4a	Receiver reports lost/stolen/rediscovered receiver device: suspend/resume keys	n/a	Suspended/ Resumed
R-DOT-4b	Receiver reports lost/stolen receiver device: remote wipe keys	n/a	Terminated
R-DOT-5a	Security breach on sender device is detected	Terminated	Terminated
R-DOT-5b	Security breach on receiver device is detected	n/a	Terminated

The use cases in [Table 13-1](#) through [Table 13-4](#) are grouped into more detailed flow diagrams. [Table 13-6](#) provides the mapping. In all the flow diagrams described in Sections [13.2](#) through [13.6](#), the exchanges between vehicle and Vehicle OEM Server, device and Device OEM Server, and Vehicle OEM Server and KTS are out of scope of this specification. These exchanges are shown for illustration purposes only. The exchanges between Vehicle OEM Server and Device OEM Server are described by the relevant server APIs (see Section [17](#)). The following APIs are used in key termination procedures:

- **manageKey()** and **manageKey() Response**: See Section [17.8.2](#)
- **eventNotification()** and **eventNotification() Response**: See Section [17.9.1](#)

Note: Only relevant parameter(s) in the API that are specific to the particular step in the flow diagram are shown.

All keys are identified by their Digital Key identifier (key id) in the Vehicle OEM Server and Device OEM Server. Digital Key identifiers are not shown explicitly in the flow diagrams unless required for comprehension. Along with the terminationAttestation parameter, the vehicleOEMProprietaryData is also transferred if not explicitly shown in the flow diagrams. Interpretations of the keyID parameter inside of the manageKey() call for action=TERMINATE (as defined in section [17.8.2](#)), are described in [Table 13-5](#) below.

Table 13-5: keyID usage for manageKey() with action=TERMINATE

Request by	ManageKey() Parameters	KeyID Usage
Sender / Receiver Device	No RTR included, TA included	Key deleted on device, key to be deleted in vehicle
Device OEM Server	No RTR included, no TA included	key requested to be deleted in vehicle
Sender Device	Device RTR included	keyID = Sender keyID, multiple keys can be deleted in RTR
Vehicle OEM Server	Server RTR included	keyID = one of the keyIDs contained in the RTR

For full parameters of each API call, please refer to the corresponding sections.

Note: Receiver key ID is used in all the figures in this section and that the term is synonymous with Digital Key identifier of a receiver .

Data elements requiring privacy encryption (See Section [13.8](#)) are identified by “*” in the flow diagrams. The exact data elements to be encrypted are defined in Section [17](#).

Messages requiring authentication (See Section [13.8](#)) are identified by “+” in the flow diagrams. The exact messages and API calls to be signed are defined in Section [17](#).

1

Table 13-6: Detailed Key Termination Use Cases

Flow ID	Description	Use Case ID
Receiver Key Remote Termination		
REV_100	Shared key termination in vehicle	R-VT-2
REV_110	Shared key termination in owner Vehicle OEM account	R-VOT-2
REV_120	Shared key termination in receiver Vehicle OEM account	R-VOT-3
REV_130	Shared key termination on owner device natively	R-DT-4
REV_130a	Shared keys termination on owner device natively (all keys on receiver account)	R-DT-4
REV_140	Shared key termination on owner device in Vehicle OEM app	R-DT-2
REV_140a	Shared keys termination on owner device in Vehicle OEM app (all keys on receiver account)	R-DT-2
REV_150	Shared key termination based on expiry date of the key	R-VOT-5
REV_160	Shared key termination by Device OEM (device security issue)	R-DOT-5b
REV_170	Shared key termination due to remote wipe of device	R-DOT-4b
REV_180	Shared key termination due to deletion of personal data in receiver Vehicle OEM account	R-VOT-10
Shared Key Local Deletion		
REV_200	Shared key termination on receiver device natively	R-DT-6
REV_210	Shared key termination on receiver device in Vehicle OEM app	R-DT-5
REV_220	Shared key termination due to local wipe of device	R-DOT-2
Shared Key Remote Suspension		
REV_300	Shared key temporary suspension in device lost mode in Device OEM account	R-DOT-4a
REV_310	Shared key resume from device lost mode in Device OEM account	R-DOT-4a
Owner Key Remote Termination		
REV_400	Owner key deletion in vehicle UI (change device)	R-VT-1, R-VT-3
REV_410	Owner key termination by Device OEM (security issue)	R-DOT-5a
REV_420	Owner key termination due to remote wipe of device	R-DOT-3b
Owner Key Local Deletion		
REV_500	Owner key termination on owner device natively (deletion of pass for Digital Key)	R-DT-3
REV_510	Owner key termination on owner device in Vehicle OEM app	R-DT-1
REV_520	Owner key termination due to local wipe of device	R-DOT-1
Owner Key Remote Suspension		

Flow ID	Description	Use Case ID
REV_600	Owner key temporary suspension in device lost mode in Device OEM account	R-DOT-3a
REV_610	Owner key resume from device lost mode in Device OEM account	R-DOT-3a
Vehicle Unpairing		
REV_700	Unpairing in vehicle UI (sale of vehicle)	R-VT-4
REV_710	Unpairing on owner device in Vehicle OEM app	R-DT-7
REV_720	Unpairing in owner Vehicle OEM account (sale of vehicle)	R-VOT-11
REV_730	Unpairing based on cancellation or expiry date of the Digital Key	R-VOT-4
REV_740	Unpairing by Vehicle OEM (vehicle stolen)	R-VOT-6
REV_750	Unpairing by Vehicle OEM (security breach in vehicle)	R-VOT-8
REV_760	Unpairing by Vehicle OEM (garage service process)	R-VOT-7
REV_770	Unpairing due to deletion of personal data in owner Vehicle OEM account	R-VOT-9
Vehicle Unbinding (leads to unpairing)		
REV_800	Unbinding of vehicle from owner account in vehicle UI (unpairing, owner sells vehicle)	R-VT-4
REV_810	Unbinding of vehicle from owner account in owner Vehicle OEM account (unpairing, owner sells vehicle)	R-VOT-11
REV_820	Unbinding of vehicle from owner account in Vehicle OEM app (unpairing, owner sells vehicle)	R-DT-7

13.2 Shared Key Remote Termination

All scenarios in which the shared key termination process is started from a device other than the receiver's own device are considered as remote termination cases. The common behavior is that the termination request is issued to the Vehicle OEM Server, which then starts the termination process on the vehicle side.

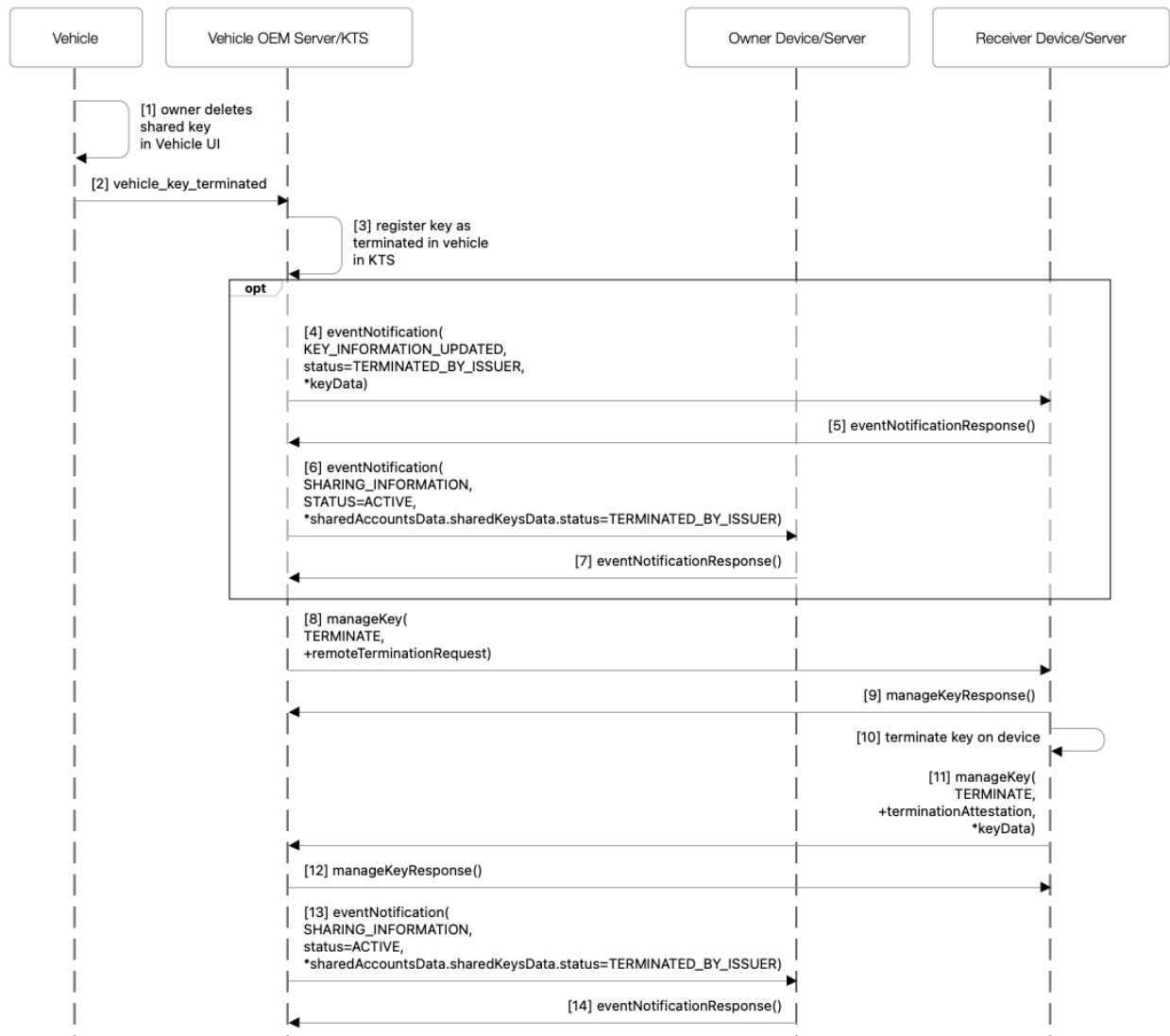
For remote termination, the key is considered as terminated when it is deleted in the vehicle.

13.2.1 REV_100: Shared key termination in vehicle

If the owner deletes a Shared key in the vehicle, no fade-out is required. The vehicle notifies the devices about the successful deletion via the Vehicle OEM Server (REV_100).

1

Figure 13-1: REV_100: Shared Key Termination in Vehicle



2

3 Alternatively, it may be required for the vehicle to request online deletion. In this case, the flow
 4 is similar to remote termination from the Vehicle OEM Server. Devices are informed about the
 5 deletion request and the successful deletion (REV_100a)



3 The eventNotification (IN_TERMINATION) should be sent in all cases so that the device side
4 does not see any difference between both options.

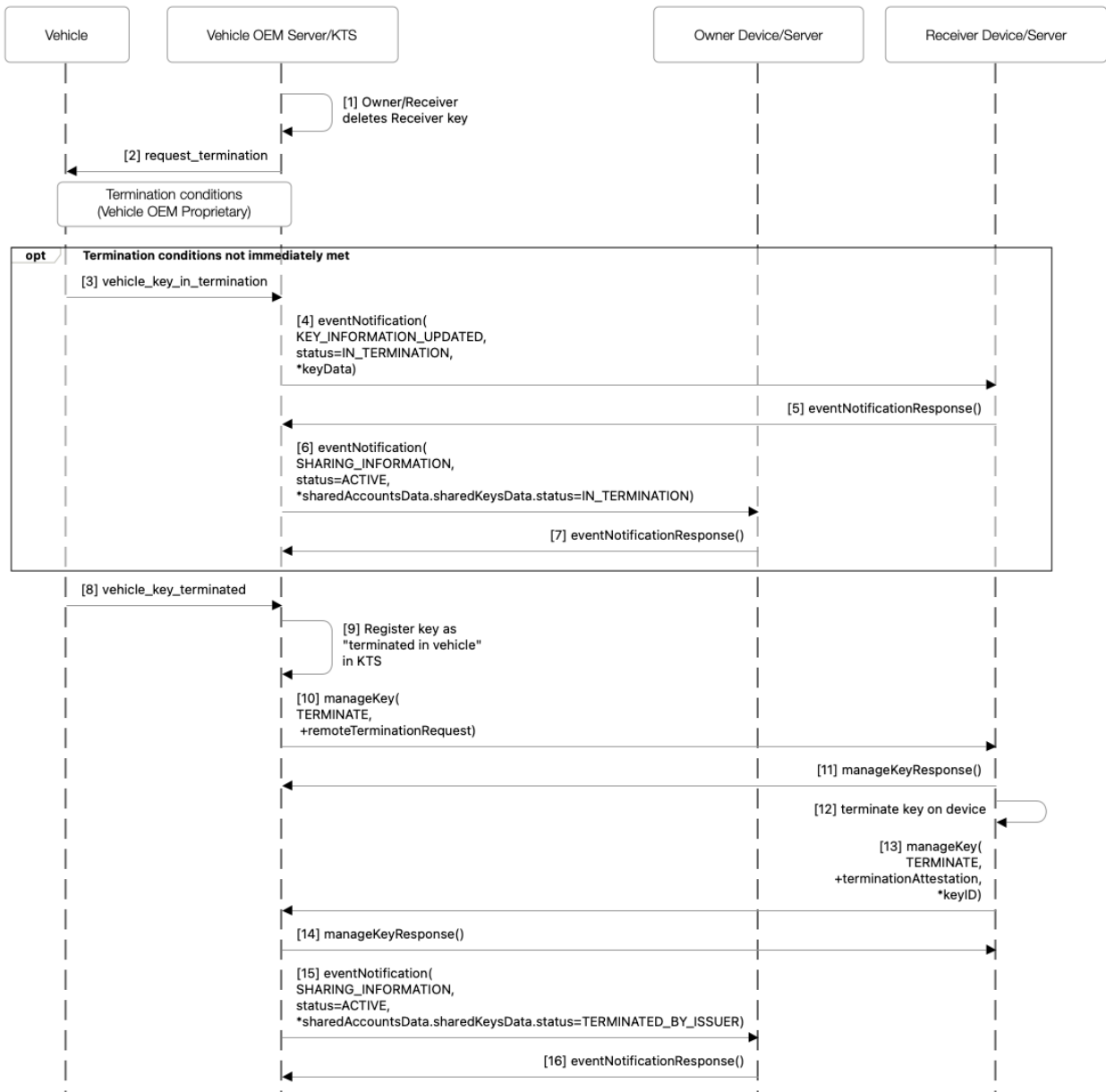
5 13.2.2 REV_110: Shared key termination in owner Vehicle OEM account

6 13.2.3 REV_120: Shared key termination in receiver Vehicle OEM account

7 The same flow shall be executed when either the owner deletes the Shared key from the owner's
8 Vehicle OEM account or the receivers delete their own key from their Vehicle OEM account.

9 This is shown in [Figure 13-3](#).

1 *Figure 13-3: REV_110/120: Shared Key Termination in Owner/Receiver Vehicle OEM Account*

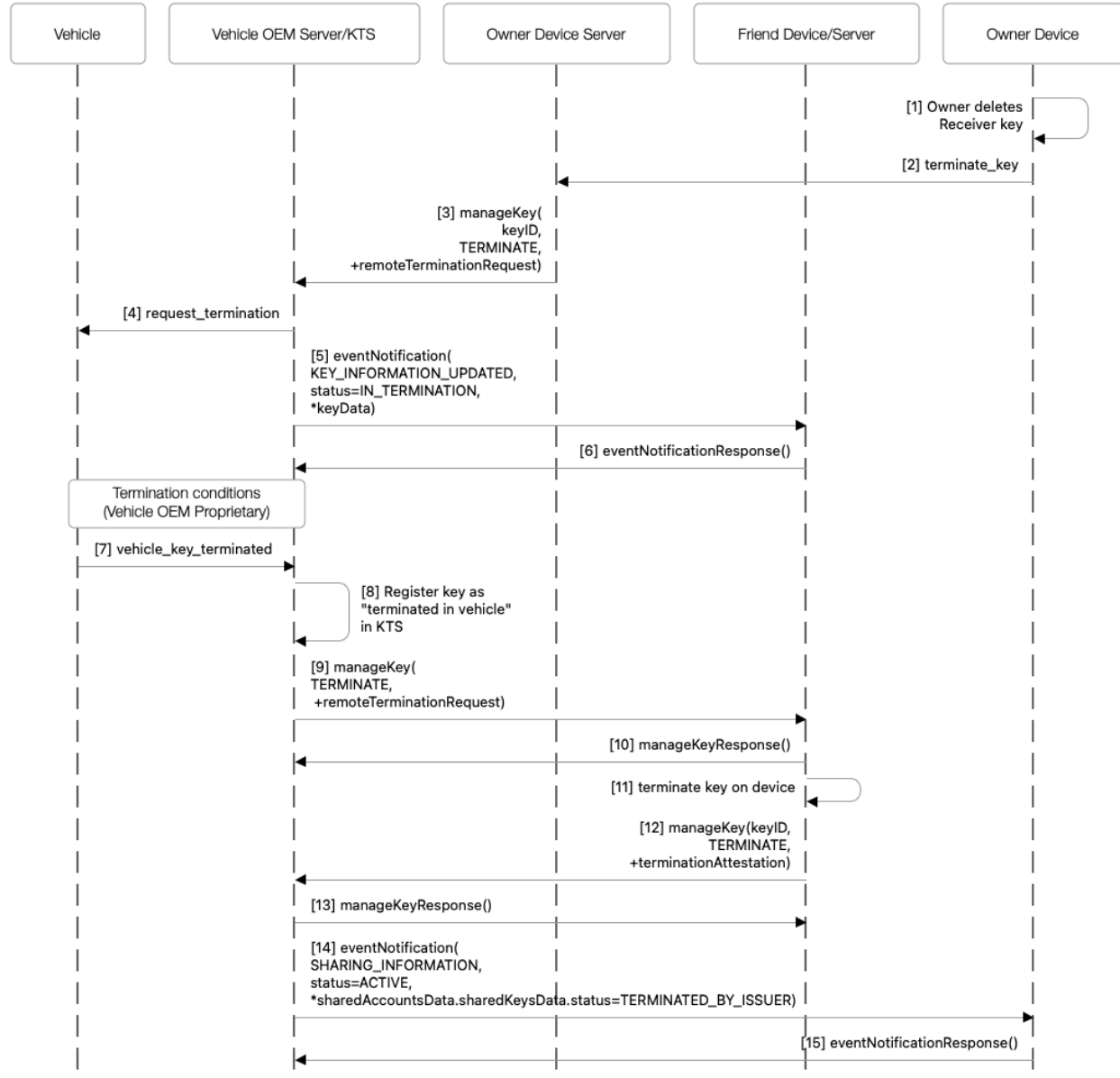


2

13.2.4 REV_130: Shared key termination on owner device natively

13.2.5 REV_140: Shared key termination on owner device in Vehicle OEM app

Figure 13-4: REV_130a/140a: Shared Key Termination on Owner Device Natively/in Vehicle OEM App



In some cases, the owner may terminate a Shared key using one of the following methods:

1. Natively through the device operating system (REV_130, see figure 13-4)
2. Using vehicle OEM apps installed on the device which leverage one or more APIs offered by the device operating system (REV_140, see figure 13-4)
3. Using vehicle OEM apps installed on the device which contact the Vehicle OEM server directly. (REV_110)

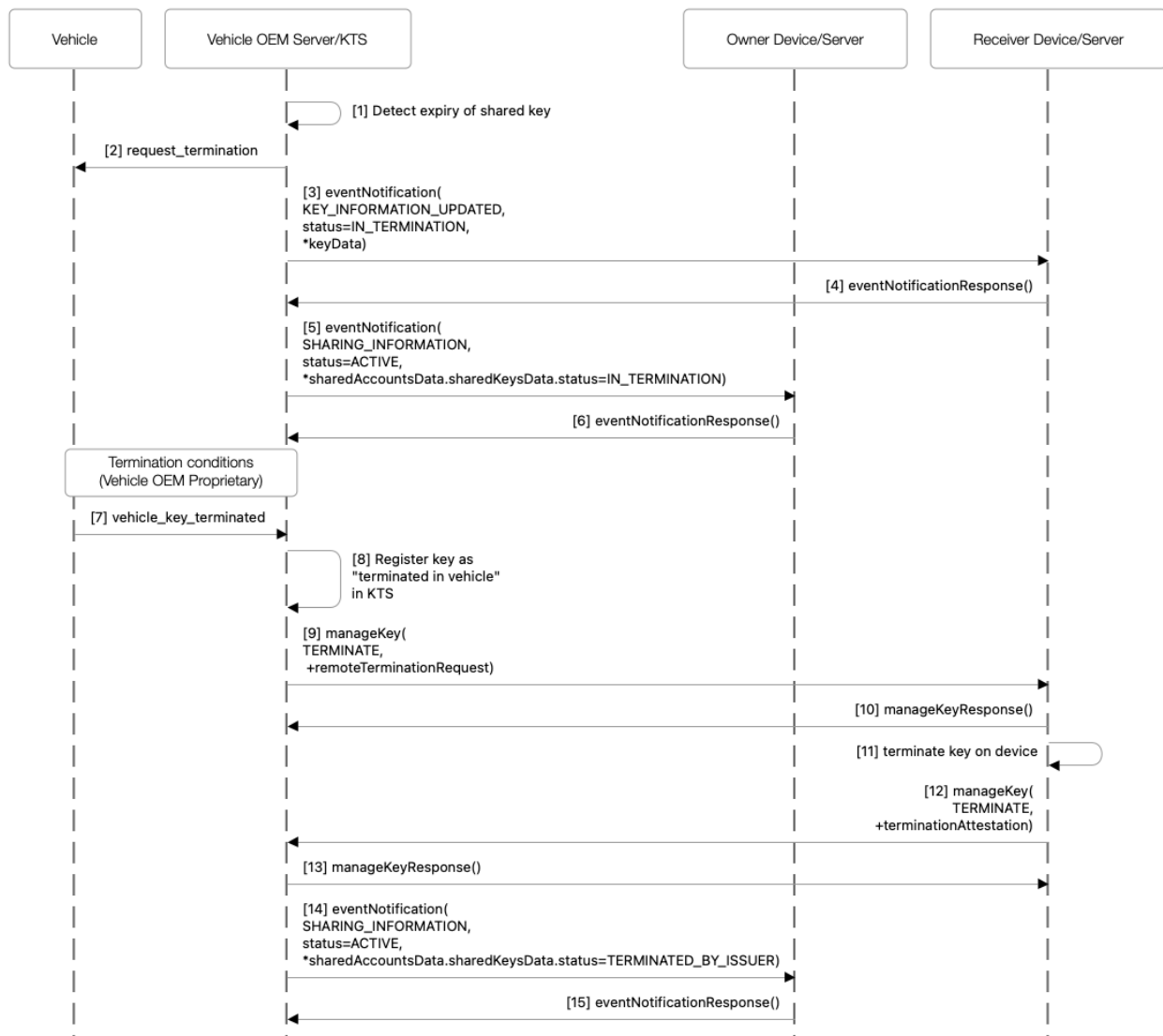
If multiple devices on the receiver's device OEM account contain a key for the vehicle, then all should be deleted by the vehicle OEM server based on the remote termination request sent by the sender device.

13.2.6 REV_150: Shared key termination based on expiry date of the key

If the vehicle or the Vehicle OEM Server detects that a Shared key has expired, the key is deleted from the vehicle in the same way as if it were deleted in the vehicle or the Vehicle OEM Server by the owner.

The flow is similar to REV_120: Shared key termination in receiver Vehicle OEM account (see Section 13.2.3).

Figure 13-5: REV_150: Shared Key Termination Based on Expiry Date of the Key



The flow depicting Vehicle detection of an expired Shared key is shown in Figure 13-6 and is similar to REV_100: Shared Key Termination in Vehicle (Figure 13-1)

1

Figure 13-6: Vehicle Detects Expiry of Shared Key



2

13.2.7 REV_160: Shared key termination by Device OEM (device security issue)

When a security issue on a device or group of devices is detected, the Device OEM removes the Digital Keys based on the nature of the security issue. The details are out of scope of this specification.

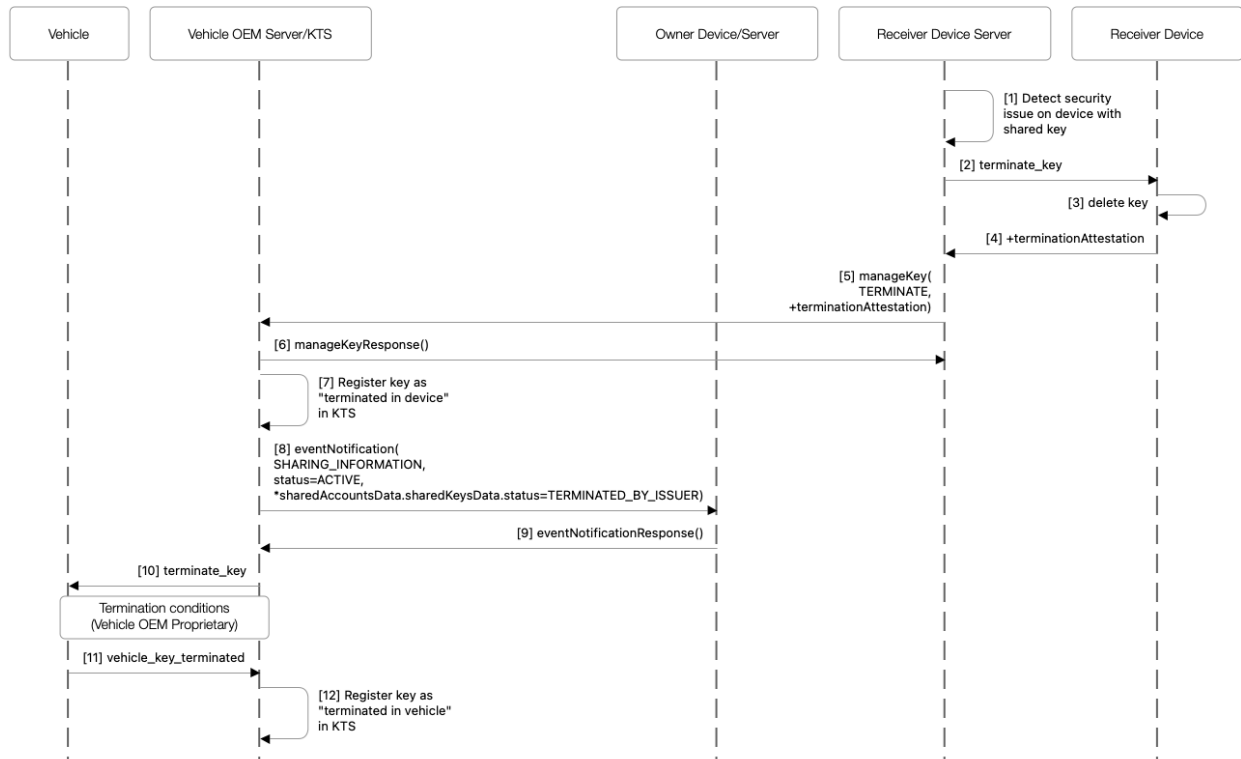
In those cases, the keys may be removed immediately from the device, depending on the security issues.

The termination attestation and the Vehicle OEM proprietary data from the device are provided to the KTS.

10

1

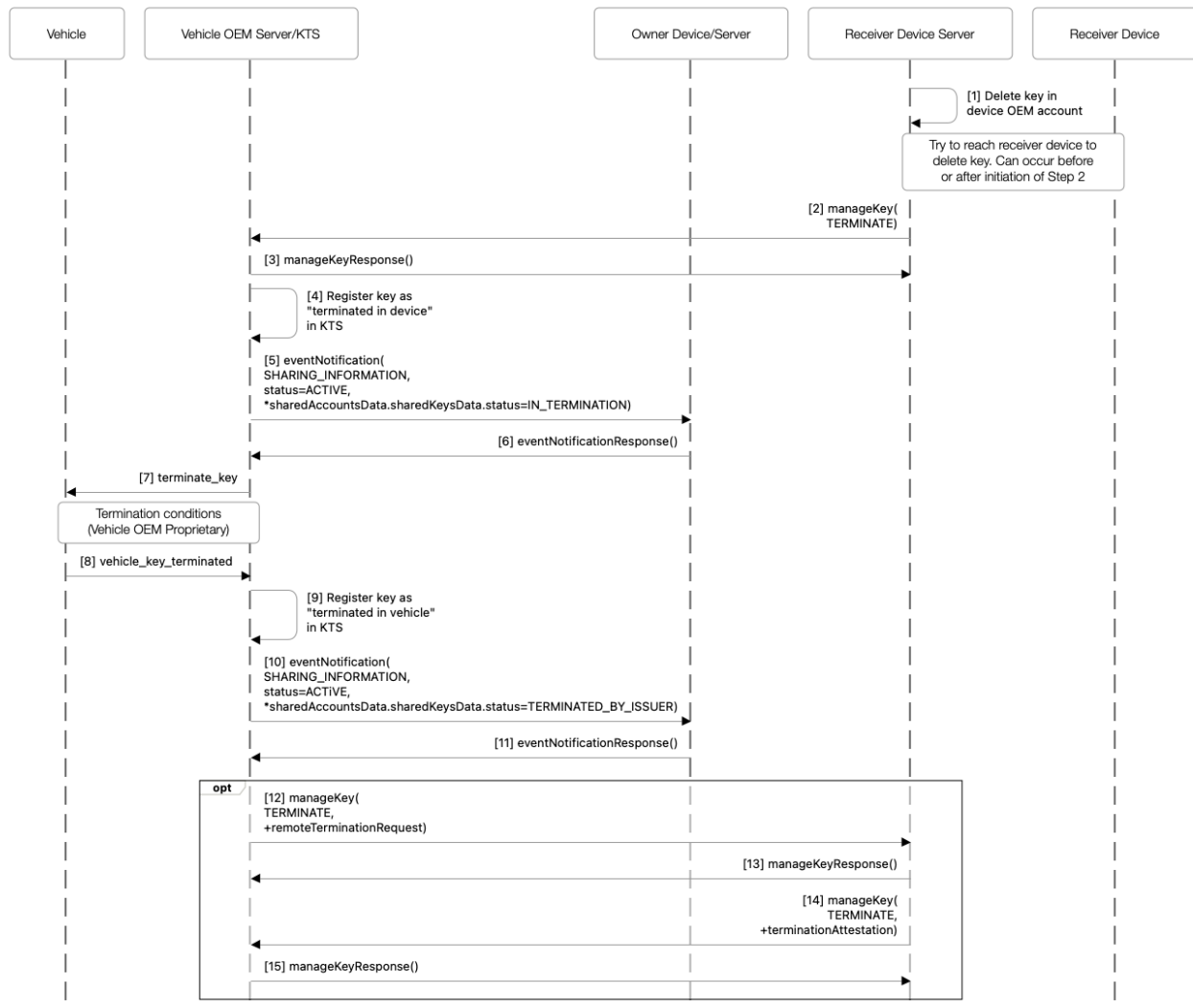
Figure 13-7: REV_160: Shared Key Termination by Device OEM (Device Security Issue)



2

3 When the device cannot be reached, the Vehicle OEM Server shall still be notified immediately,
4 as shown in [Figure 13-8](#).

1 *Figure 13-8: REV_160a: Shared Key termination by Device OEM and Receiver device is Offline*



2
3 The termination attestation and the Vehicle OEM proprietary data from the device may be
4 provided to the KTS if the device is online.

5 13.2.8 REV_170: Shared key termination due to remote wipe of device

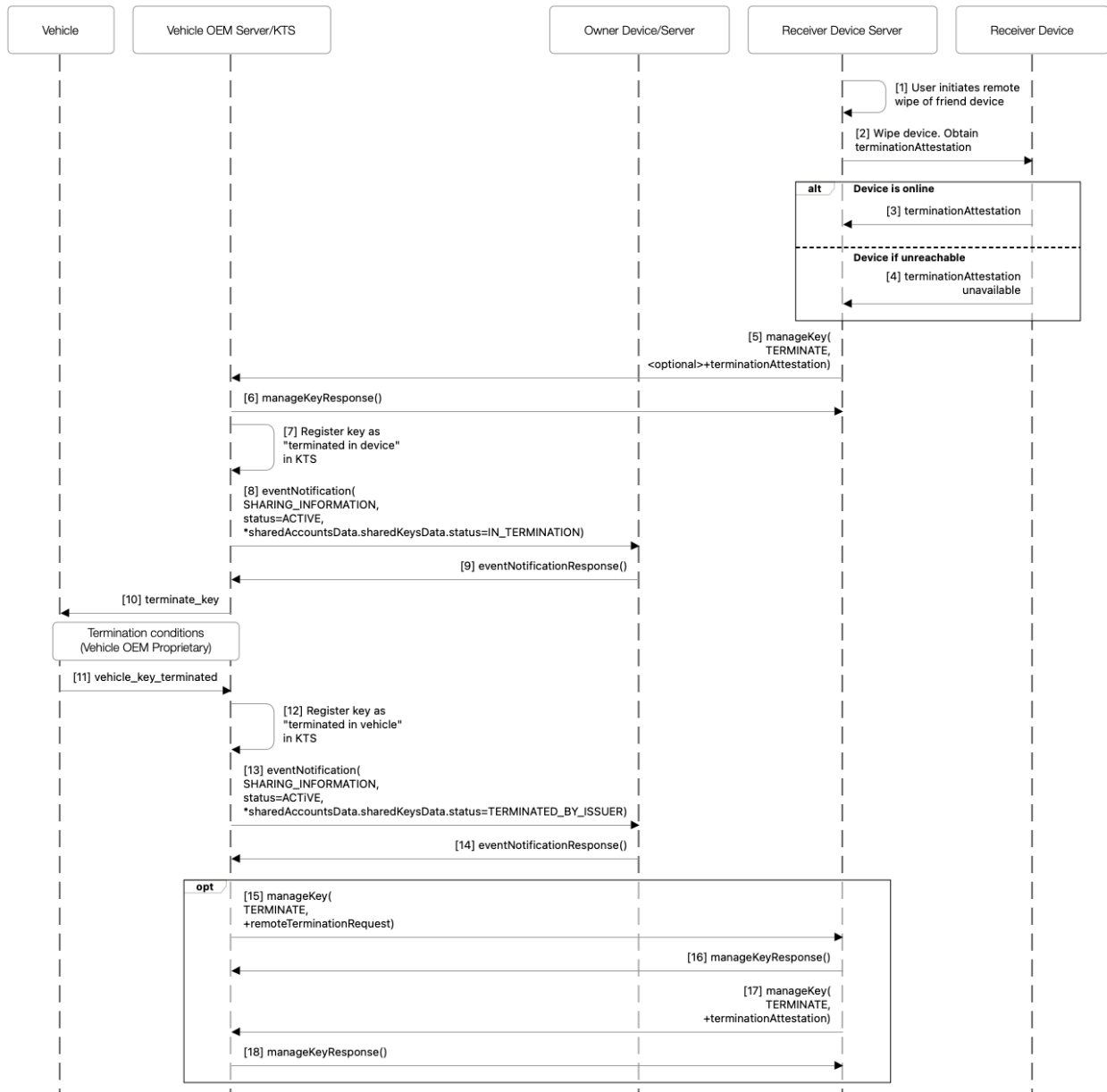
6 When a receiver decides to wipe his/her device remotely, the Device OEM deletes the Digital
7 Keys on the involved devices. The details are out of scope of this specification.

8 In those cases, the keys are deleted immediately from the receiver's device. Vehicle and Vehicle
9 OEM termination conditions and fade-out do not apply.

10 *Note:* In the case of a remote wipe of a device, the device may include terminationAttestation
11 and vehicleOEMProprietaryData in manageKey() in Step 3, as shown in [Figure 13-9](#).

1

Figure 13-9: REV_170: Shared Key Termination Due to Remote Wipe of Device



2

3 In some cases, the receiver may terminate the Shared key using one of the following methods:

- 4 1. Natively through the device operating system (REV_200, see Figure 13-10)
- 5 2. Using vehicle OEM apps installed on the device which leverage one or more APIs
- 6 offered by the device operating system (REV_210, see Figure 13-10)
- 7 3. Using vehicle OEM apps installed on the device which contact the vehicle OEM server
- 8 directly (REV_120)

13.2.9 *REV_180: Shared key termination due to deletion of personal data in receiver Vehicle OEM account*

When the receiver has a Vehicle OEM account, the shared key is linked with the account. A request to delete the personal data from the account should lead to removal of the key from the vehicle, similar to a key deletion as described in “REV_120: Shared key termination in receiver Vehicle OEM account” (see Section [13.2.3](#)).

13.3 Shared Key Local Termination

All scenarios in which the Shared key termination process is started on the receiver’s own device are considered as local termination cases.

The common behavior is that the termination request should be executed immediately on the receiver device. Depending on the connectivity situation, the termination attestation and the Vehicle OEM proprietary data may be sent immediately to the KTS, delayed, or not sent at all.

In cases where the Vehicle OEM Server is not aware of the termination, termination attempt, or loss of the device, the owner or receiver (if applicable) should remove the key in the vehicle.

Fade-out or termination preconditions do not apply when termination attestation and the Vehicle OEM proprietary data will be transmitted to the Vehicle OEM Server.

The owner device should be notified about successful registration of the termination. The receiver and owner device UI should mark the corresponding key representation as “terminated.”

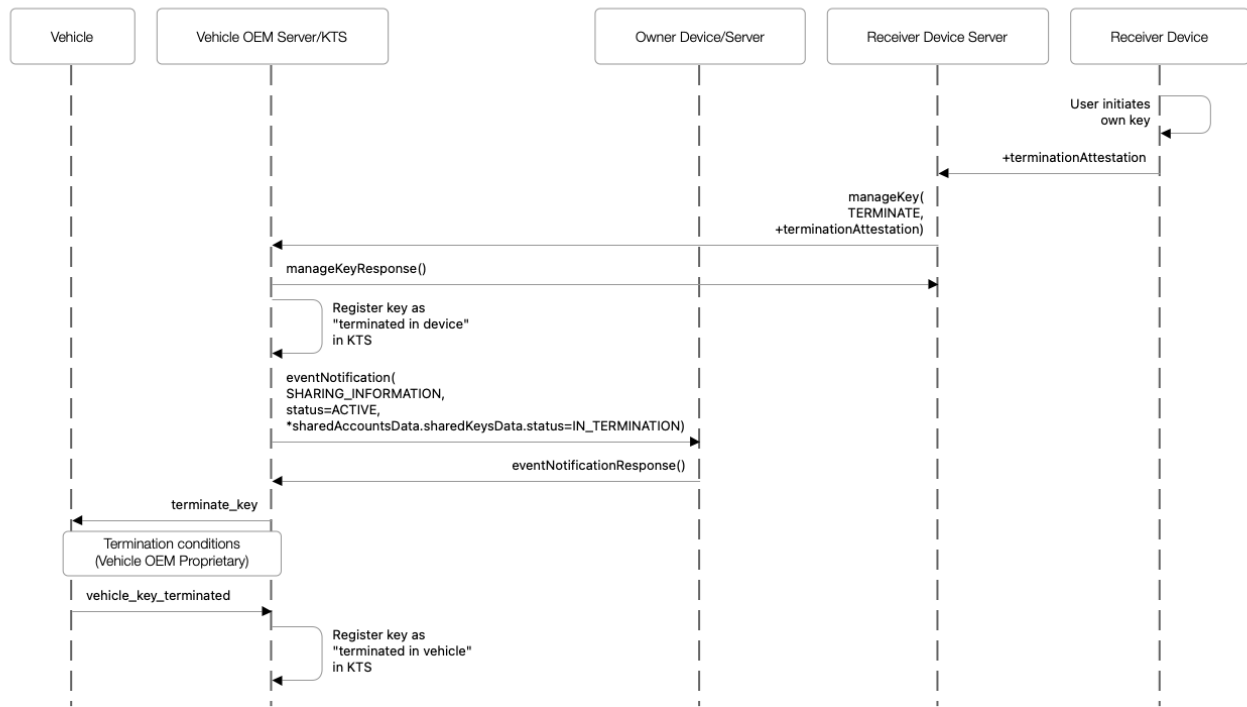
13.3.1 *REV_200: Shared key termination on receiver device natively*

13.3.2 *REV_210: Shared key termination on receiver device in Vehicle OEM app*

If the key is deleted on the receiver device, the termination attestation and the Vehicle OEM proprietary data shall be sent to the receiver server as soon as the device is online.

The server flow is exactly the same for native and Vehicle OEM app-initiated deletion.

1 *Figure 13-10: REV_200/210: Shared Key Termination on Receiver device Natively/in Vehicle OEM App*



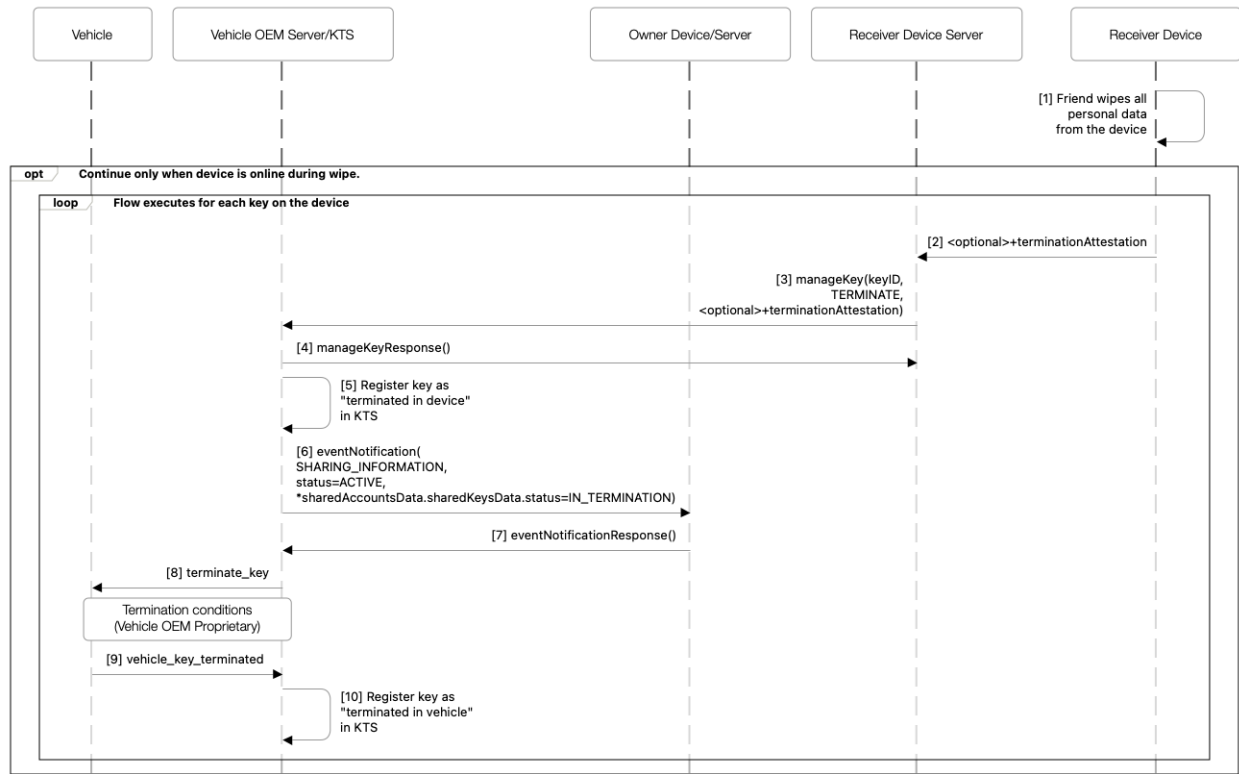
3 The Vehicle OEM indicates to the vehicle that the key can be terminated immediately when the
4 device key is registered in the KTS as “terminated on device.”

5 13.3.3 REV_220: Shared key termination due to local wipe of device

6 If the device is wiped locally, the termination attestation and the Vehicle OEM proprietary data
7 is sent to the receiver server on a best-effort basis.

1

Figure 13-11: REV_220: Shared Key Termination Due to Local Wipe of Device



2

- 3 The Vehicle OEM indicates to the vehicle that the key can be terminated immediately when the
 4 device key is registered in the KTS as “terminated on device.”
 5 For each Shared key present on the device, the flow shown in Figure 13-11 shall be executed.
 6 The owner key termination due to device wipe is described in Section [13.5.6](#).

7 13.4 Shared Key Remote Suspend/Resume

8 Remote suspend and resume occur when the device is set to lost or stolen mode in the Device
 9 OEM Server and when the lost/stolen mode is ended.

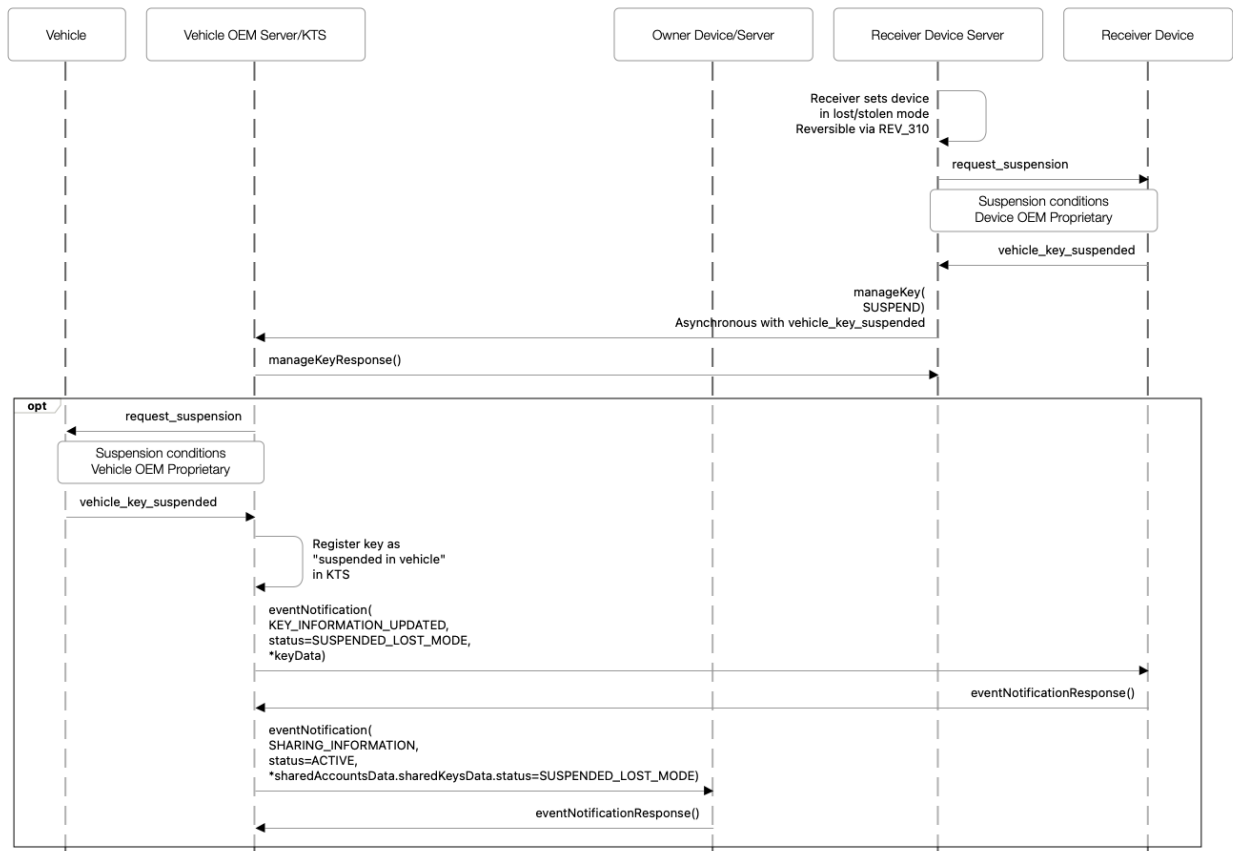
10 If the device is reachable, the keys are suspended immediately on the device. The suspended
 11 status is reversible; keys may be resumed through the Device OEM Server or locally on the
 12 device. No cryptographic attestation is available for a key suspended and resumed on the device.

13 13.4.1 REV_300: Shared key temporary suspension in Device OEM account

14 Suspension of a Digital Key triggered by the Device OEM account is used when the device mode
 15 is set to lost or stolen. The Device OEM Server attempts to suspend the key on the device and, in
 16 parallel, the Vehicle OEM Server is notified to suspend the keys in the vehicle, as shown in
 17 [Figure 13-12](#).

1

Figure 13-12: REV_300: Shared Key Suspension by Device OEM Account



2

3 The suspension on the device is not tracked in the KTS. The suspension of the key in the vehicle
4 may be tracked by the KTS.

5 manageKey SUSPEND command communicates the intent of the user to suspend the key to the
6 KTS, but not the suspension status in the device.

7 The Vehicle OEM notifies the owner and receiver when the key is successfully suspended in the
8 vehicle.

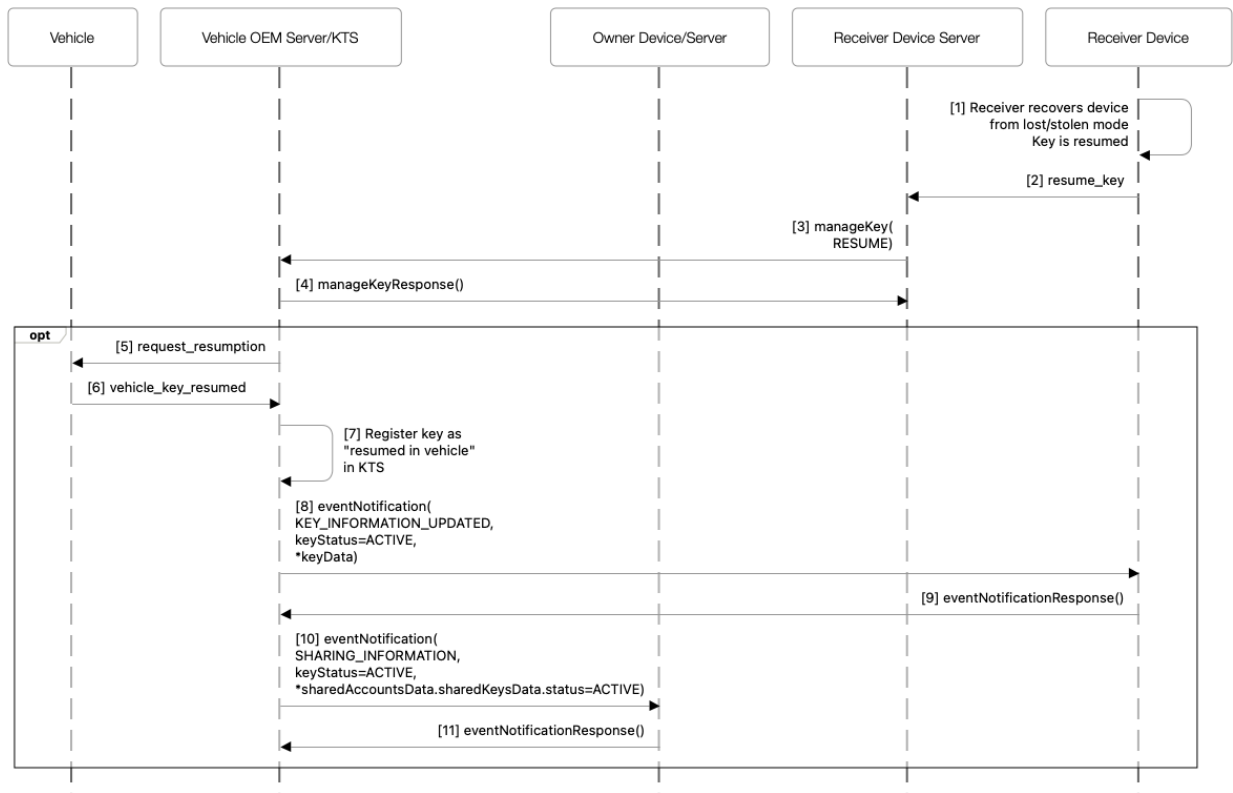
9 Regardless of vehicle OEM server support of remote suspend/resume or not, the Vehicle OEM
10 server shall response with manageKeyReponse (statusCode 200) if it receives the manageKey()
11 Request successfully.

12 13.4.2 REV_310: Shared key resume in Device OEM account

13 In the case where the Vehicle OEM Server supports remote suspend/resume, the suspended key
14 may be resumed in the Device OEM account or on the device itself if it has been recovered, as
15 shown in [Figure 13-13](#). The receiver device OEM Server shall only send manageKey(RESUME)
16 or transmit a resume attestation in the receiver private mailbox if it has previously received
17 eventNotification(SUSPENDED).

1

Figure 13-13: REV_310: Shared Key Resume in Device OEM Account or on Device



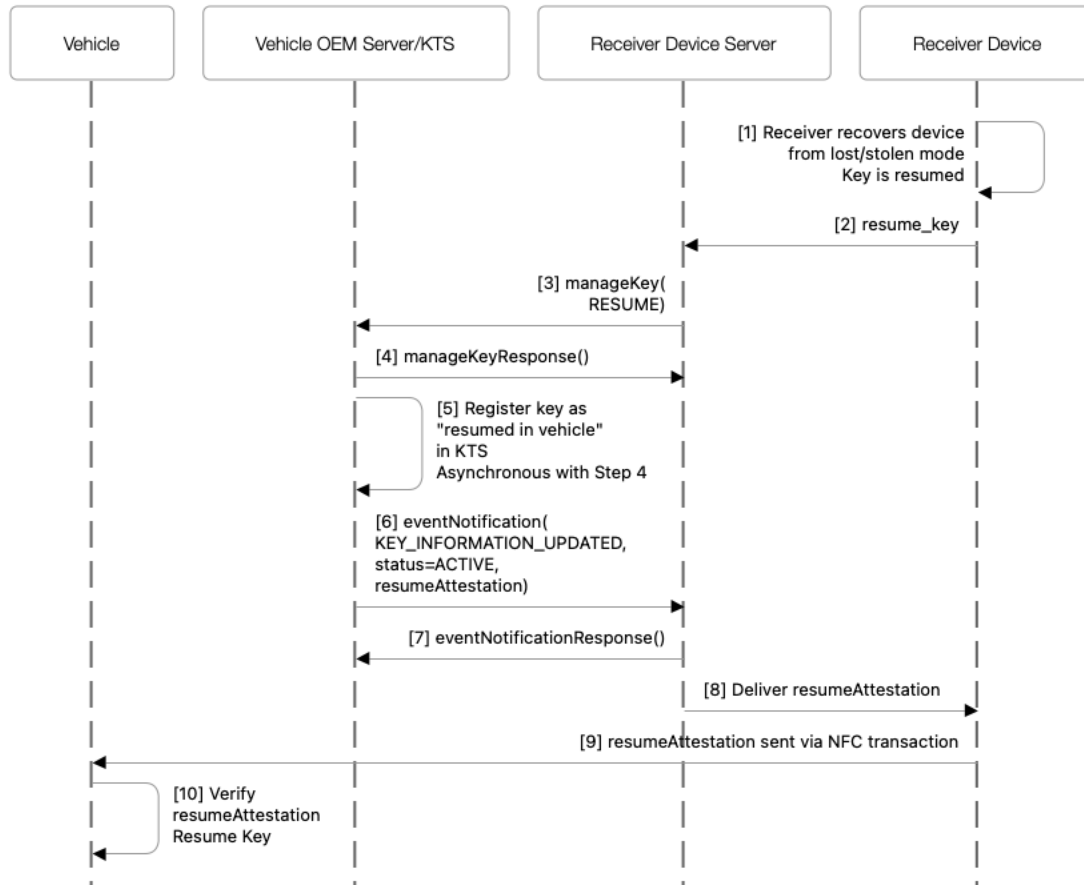
2

3 The device shall be online to resume the key and notify the Vehicle OEM.

4 If the key is previously tracked as “suspended in vehicle,” the key shall be tracked as “resumed
5 in vehicle” before the vehicle resumes the key to avoid a state in which the key is resumed but
6 still tracked as suspended.

1

Figure 13-14: REV_310a: Shared Key Resume Using ResumeAttestation



2

3 When the vehicle is moved into an offline area between suspend and resume, the receiver shall
4 be able to provide a resume attestation via NFC transaction to the vehicle, as shown in [Figure](#)
5 [13-14](#).

6 13.4.3 Resume attestation

7 When a Digital Key has been suspended in a vehicle, it shall be possible to bring a Resume
8 Attestation using the private mailbox of an owner's or receiver's phone in a similar way as it is
9 done for the First New Key Transaction.

10 The Resume Attestation may be used when the vehicle is offline and the owner Digital Key (see
11 Section [13.5.7](#)) or the Shared Digital Key (see Section [13.4.1](#)) is in a suspended state.

12 The device shall add the tag 7F61_h and length fields to the resume-Attestation (see [Table 13-7](#))
13 and store the TLV structure in the KeyAtt field of the private mailbox (see Table 4-3). The
14 Resume Attestation shall be written by the Digital Key framework in the private mailbox at
15 offset KEY_ATT_OFFSET and the DEVICE_KEY_ATT_BIT signaling bit (see Table 4-2) shall
16 be set to 1.

17 During a transaction, the vehicle detects the DEVICE_KEY_ATT_BIT and consumes the
18 attestation. The vehicle then clears (set to 0) the DEVICE_KEY_ATT_BIT and clears the
19 mailbox area at offset KEY_ATT_OFFSET.

On reception of a valid Resume Attestation, the vehicle resumes the designated key or does nothing if the Digital Key is already resumed.

Implementation Notes:

If the suspend/resume feature is supported by the Vehicle OEM, the Vehicle OEM must ensure that the Resume Attestation cannot be replayed using a proprietary method.

Table 13-7: Resume Attestation

Tag	Length	Description	Field is	Domain Version
7F61 _h	variable	resumeAttestation (see Section 17.9.1)		V-OD-FW
		vehicle OEM proprietary	mandatory	N/A

13.5 Owner Key Remote Termination

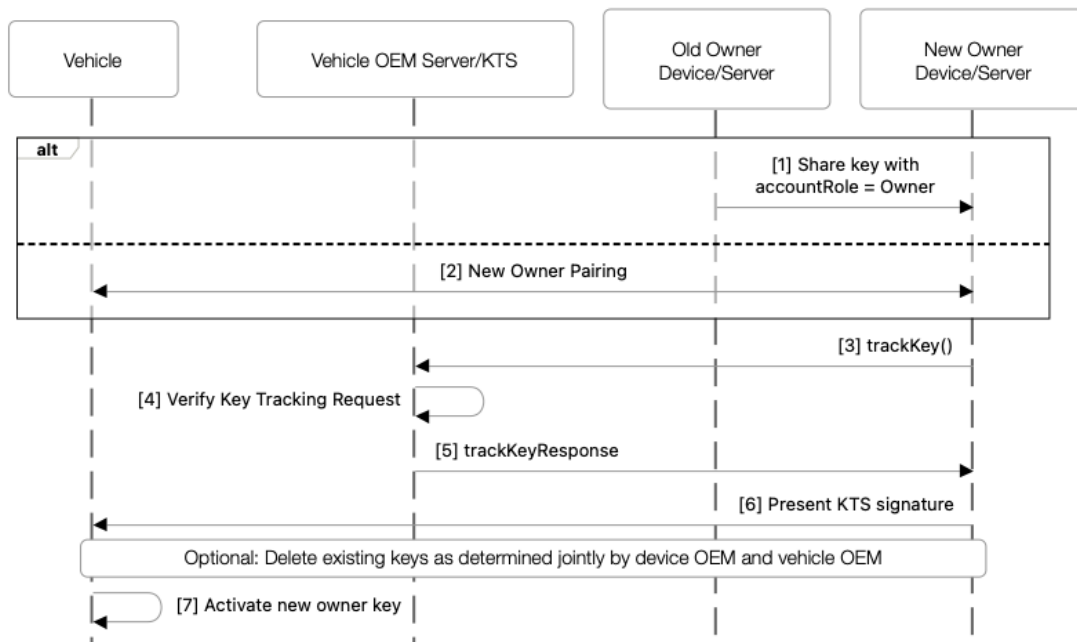
Owner key remote termination shall be handled very similar to Shared key remote termination as described in Section 13.2. The receiver device is not informed about owner key termination. Owner key termination does not affect the functionality of Shared keys for the same vehicle.

13.5.1 REV_400: Change owner device

A new owner key may be created by sharing a new key with accountRole = (owner) to a new device. By using this method, owner pairing is not required to be executed by the new owner device.

If a new owner device is paired with the vehicle, existing keys may be deleted as determined jointly by the device OEM and vehicle OEM

Figure 13-15: REV_400: Owner Key Deletion in Vehicle UI (Change of Device)



Alternatively, it may be required for the vehicle to request online deletion. In that case, devices are informed about the termination request and the successful termination (See [Figure 13-16](#)).

Figure 13-16: REV_400a: Owner Key Deletion in Vehicle UI (Change of Device)



Note: REV_400 and REV_400a are similar to flow REV_100 and REV_100a, respectively, but without involvement of the receiver device.

13.5.1.1 *Transmission of the shared key information to the new owner device*

After the owner device change, information related to the shared keys associated with the owner Digital Key is sent to the new owner device in Step 22 of [Figure 13-15](#) using trackKey() Response (see Section [17.7.3.3](#)).

13.5.2 *REV_410: Owner key termination by Device OEM (security issue)*

This flow corresponds to REV_160 with the owner device instead of the receiver device. In addition, all Shared keys are terminated, and the receiver devices are notified.

13.5.3 *REV_420: Owner key termination due to remote wipe of device*

This flow corresponds to REV_170 with the owner device instead of the receiver device.

13.5.4 *REV_500: Owner key termination on owner device natively (delete pass for Digital Key)*

This flow corresponds to REV_200 with the owner device instead of the receiver device.

13.5.5 *REV_510: Owner key termination on owner device in Vehicle OEM app*

This flow corresponds to REV_210 with the owner device instead of the receiver device.

13.5.6 *REV_520: Owner key termination due to local wipe of device*

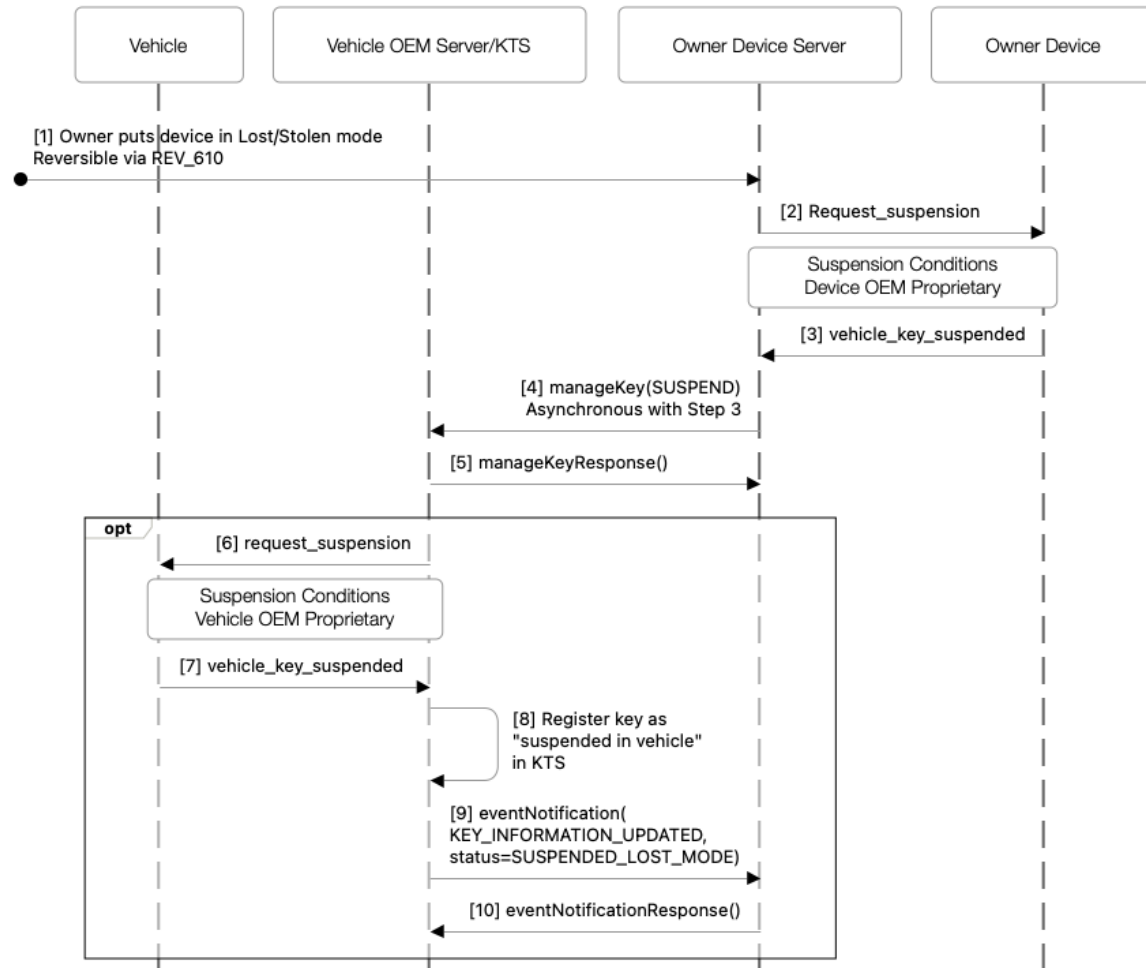
This flow corresponds to REV_220 with the owner device instead of the receiver device.

13.5.7 *REV_600/610: Owner key temporary suspension in device lost mode in Device OEM account*

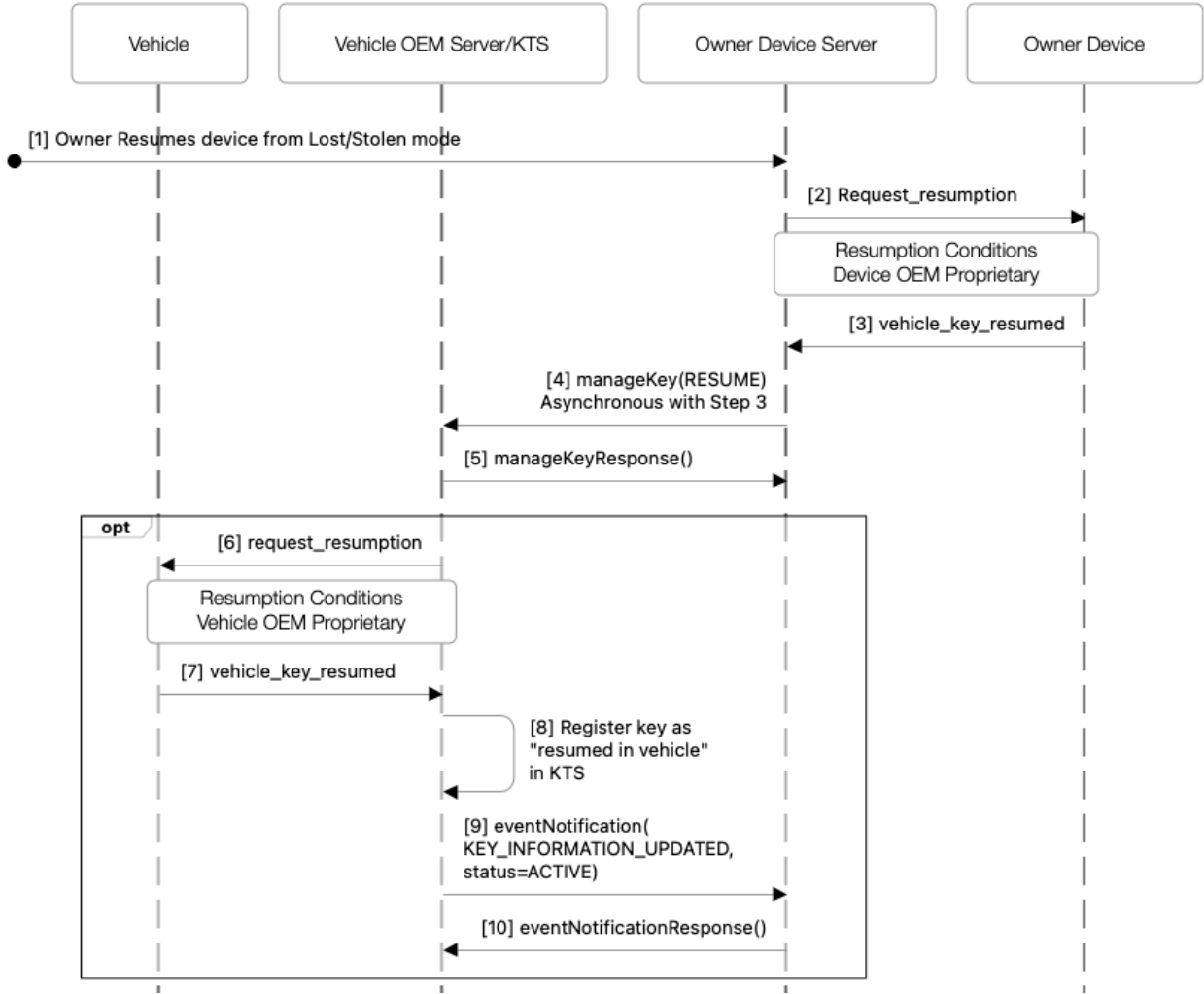
This flow corresponds to REV_300 and REV_310 with the owner device instead of the receiver device.

On owner key SUSPEND/RESUME, manageKey() Response from Vehicle OEM Server shall contain Shared Key Data for all active Shared keys associated with the Vehicle.

Figure 13-17: REV_600: Owner key suspension due to device reported lost/stolen



1 *Figure 13-18: REV_610: Owner key resumption after device reported lost/stolen*



2

3 **13.6 Owner Device Unpairing**

4 If the owner device is unpaired from the vehicle, all Shared keys shall be deleted in the vehicle

5 and the receiver devices should be notified.

6 The unpaired state shall be registered in the KTS. Key deletion on the owner and receiver

7 devices is optional but recommended.

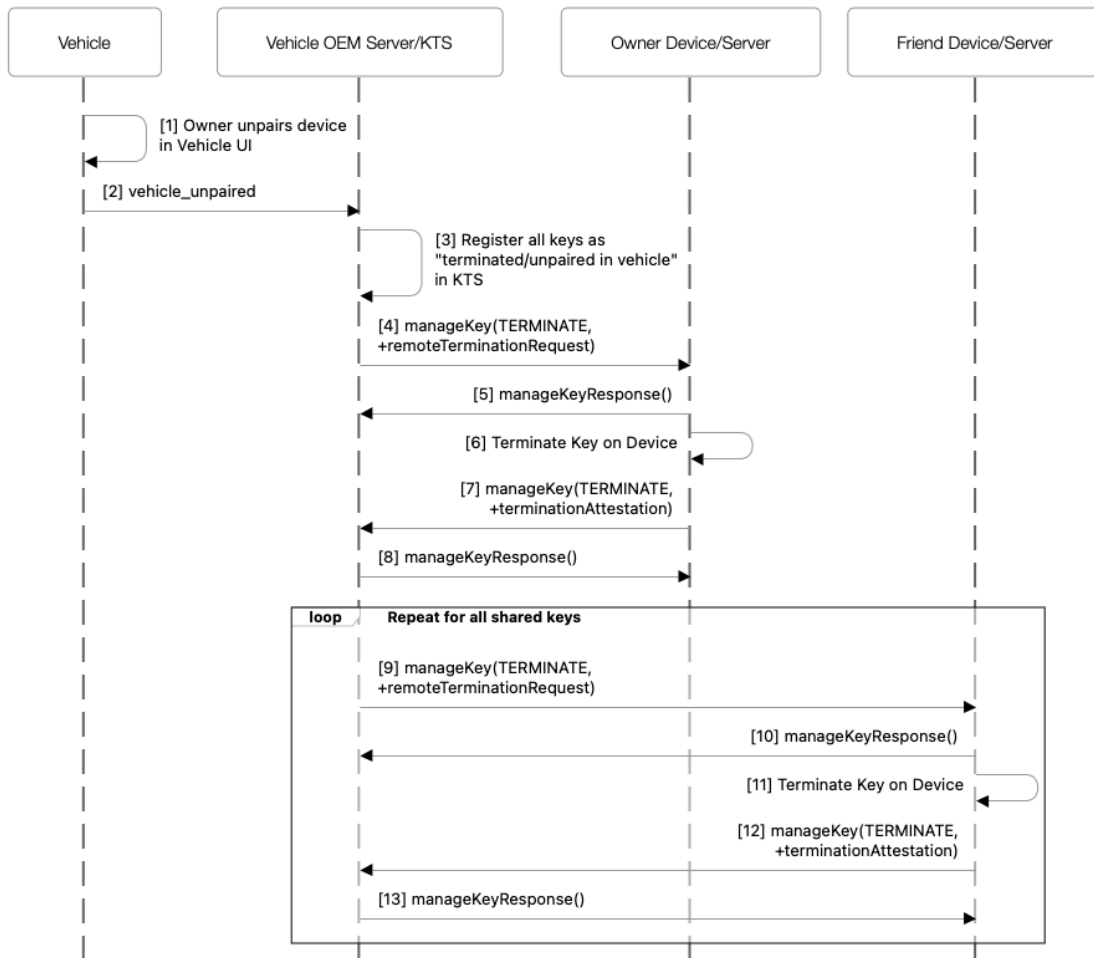
8 **13.6.1 REV_700: Unpairing in vehicle UI (sell vehicle)**

9 If unpairing is started from the vehicle, key deletion is executed under Vehicle OEM-defined

10 conditions, and the owner device is notified.

1

Figure 13-19: REV_700: Unpairing in Vehicle UI (Sale of Device)

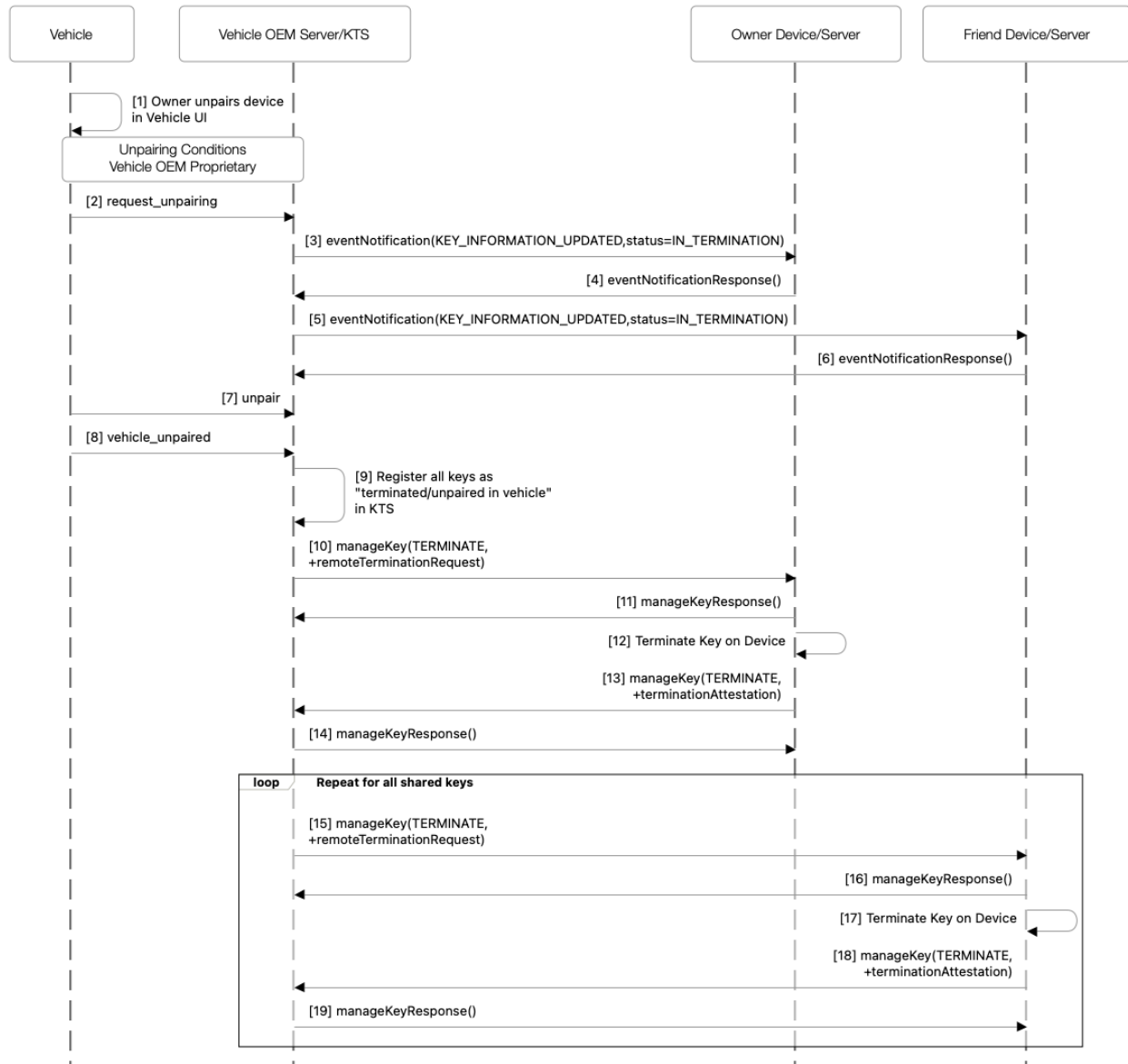


2

3 Alternatively, it may be required for the vehicle to request online unpairing. In that case, devices
 4 are informed about the termination request and the successful termination (See [Figure 13-20](#)).
 5 Note: Inability to reach owner or receiver device in Steps 3 through 6 does not stop the process

1

Figure 13-20: REV_700a: Unpairing in Vehicle UI (Vehicle Required to be Online)



2

3 13.6.2 REV_710: Unpairing on owner device in Vehicle OEM app

4 If the owner device is unpaired through the Vehicle OEM app, the app shall not start the
 5 unpairing process on the device. The app shall notify the Vehicle OEM Server about the
 6 unpairing request, and the Vehicle OEM Server shall execute the unpairing process. Note:
 7 Inability to reach receiver device in Steps 5 through 6 does not stop the process.

1

Figure 13-21: REV_710: Unpairing in Vehicle OEM App on Owner Device



2

3 13.6.3 REV_720: Unpairing in owner Vehicle OEM account (sale of vehicle)

4 This flow is similar to REV_710 with the Vehicle OEM Server triggering the unpairing process.

5 13.6.4 REV_730: Unpairing based on cancellation or expiry date of the Digital Key service

6 This flow is similar to REV_710 with the Vehicle OEM Server triggering the unpairing process.

7 13.6.5 REV_740: Unpairing by Vehicle OEM (vehicle stolen)

8 This flow is similar to REV_710 with the Vehicle OEM Server triggering the unpairing process.

9 13.6.6 REV_750: Unpairing by Vehicle OEM (security breach in vehicle)

10 This flow is similar to REV_710 with the Vehicle OEM Server triggering the unpairing process.

13.6.7 *REV_760: Unpairing by Vehicle OEM (garage service process)*

This flow is similar to REV_710 with the Vehicle OEM Server triggering the unpairing process.

13.6.8 *REV_770: Unpairing due to deletion of personal data in owner Vehicle OEM account*

This flow is similar to REV_710 with the Vehicle OEM Server (account) triggering the unpairing process.

13.6.9 *REV_800: Unbinding of vehicle from owner account in vehicle UI (unpairing, owner sale of vehicle)*

This flow is similar to REV_710 with the vehicle UI triggering the unpairing process.

13.6.10 *REV_810: Unbinding of vehicle from owner account in owner Vehicle OEM account (unpairing, owner sale of vehicle)*

This flow is similar to REV_710 with the Vehicle OEM account triggering the unpairing process.

13.6.11 *REV_820: Unbinding of vehicle from owner account in Vehicle OEM app (unpairing, owner sale of vehicle)*

This flow is similar to REV_710 with the Vehicle OEM app triggering the unpairing process.

13.7 Termination in Vehicle

13.7.1 Rules

Digital Key termination on the vehicle side requires protection against replaying all or a part of an earlier key sharing process. Slot identifiers are assigned by the vehicle OEM to the key storage slots in the vehicle and are provided to each shared key in the key tracking response. The vehicle shall verify that the slot identifier in the private mailbox of the share key matches one of the empty key storage slot. The following preconditions shall apply:

- **C1:** The initial slot identifier (SlotID in the example below) values in the private mailbox shall be set to all bytes FF_h.
- **C2:** The initial Slot Identifier Bitmap (SlotID Bitmap in the example below) value is 00_h.

The vehicle shall implement the following rules:

- **Rule V1:** Accept a key addition only if the receiver slot identifier (as described in Section [4.3.1](#)) is verified to be valid for an available key storage in the vehicle.
- **Rule V2:** When a key is deleted in the vehicle (from any interface), delete the data (device PK, etc.) in the corresponding key storage and assign a new slot identifier. The vehicle OEM shall implement appropriate means to ensure replay protection. This new slot identifier is available for future Digital Key sharing and is provided by the vehicle OEM server to a new share key.

All devices shall implement the following rules:

- **Rule D1** (only applies to owner device): not applicable for online slot identifiers.
- **Rule D2:** When a key is terminated in the device locally, send the termination attestation and the Vehicle OEM proprietary data to the Vehicle OEM Server to trigger the deletion of the Digital Key in the vehicle.

13.8 Termination and Deletion of FMS Use Cases

This section describes different Fleet Management related key termination use cases. Vehicle defleeting is the removal of the vehicle from the fleet managed by the FMS and results in the deletion of the Owner key for that vehicle stored on the SBOD.

13.8.1 Key Termination Scenarios

Table 13-8: User initiated Fleet Shared key termination scenarios

Use Case ID	Trigger	Owner Key Status	Shared Key Status
FMS-RDT-1	Vehicle user terminates Shared key in vehicle OEM app	-	Terminated
FMS-RDT-2	Vehicle user terminates Shared key in native app	-	Terminated
FMS-RDT-3	Vehicle user terminates Shared key in FMS app	-	Terminated
FMS-RDT-4	Vehicle user terminates Shared key in the vehicle	-	Terminated

Table 13-9: Vehicle OEM Triggered Fleet Key Termination

Use Case ID	Trigger	Owner Key Status	Shared Key Status
FMS-VOT-1	Unpairing based on cancellation or expiry date of the Digital Key service to FMS	Terminated	Terminated
FMS-VOT-2	Terminate Shared key after expiry date (automatic)	-	Terminated
FMS-VOT-3	FMS/legal authorities reports a stolen/destroyed vehicle	Terminated	Terminated
FMS-VOT-4	Garage service process	Terminated	Terminated
FMS-VOT-5	Security breach on vehicle	Terminated	Terminated
FMS-VOT-6	Security breach on FMS (FMS not working)	Terminated	Terminated

Table 13-10: Device OEM Triggered Fleet Key Termination

Use Case ID	Trigger	Owner Key Status	Shared Key Status
FMS-RDOT-1	Receiver wipes receiver device (factory reset)	-	Terminated
R-DOT-4a	Receiver reports lost/stolen receiver device: suspend keys	-	Suspended
R-DOT-4a	Receiver key resume from device lost mode in Device OEM account	-	Suspended lost mode
R-DOT-4b	Receiver reports lost/stolen receiver device: remote wipe keys	-	Terminated
R-DOT-5b	Security breach on receiver device	-	Terminated

Table 13-11: FMS Triggered Fleet Key Termination

Use Case ID	Trigger	Owner Key Status	Shared Key Status
FMS-FMST-1	FMS requests vehicle user shared Digital Key termination (end of sharing service, etc.)	-	Terminated
FMS-FMST-2	FMS requests vehicle defleeting	Terminated	Terminated
FMS-FMST-3	Security breach on FMS (FMS not working)	Terminated	Terminated

13.8.2 Use Case/Flow Mapping

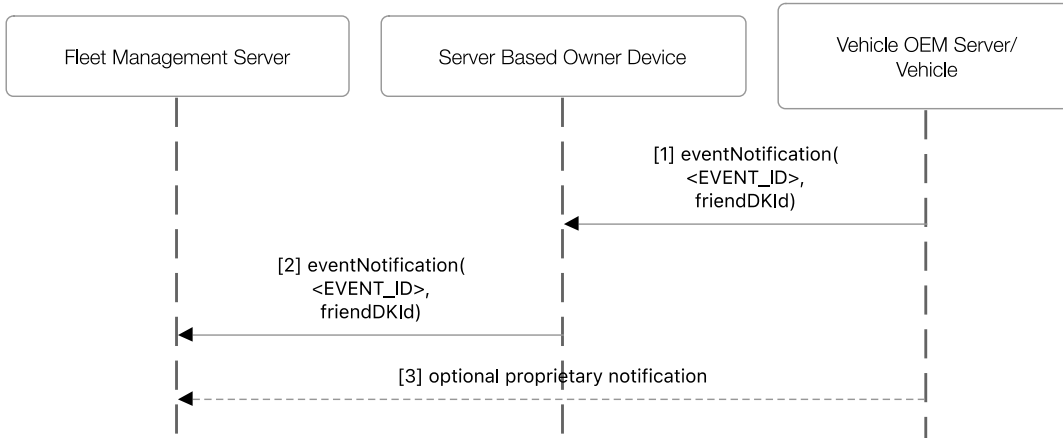
Table 13-12: FMS Flow to Use Case Mapping

Flow ID	Description	Use Case ID
Vehicle user DK suspension/termination		
REV_FMS_100	Vehicle user terminates shared Digital Key in vehicle OEM app	FMS-RDT-1
REV_FMS_110	Vehicle user terminates shared Digital Key in native app	FMS-RDT-2
REV_FMS_190	Vehicle user terminates shared Digital Key in the vehicle	FMS-RDT-4
REV_FMS_120	FMS app terminates vehicle user DK	FMS-RDT-3
REV_FMS_120	FMS terminates vehicle user DK	FMS-FMST-1
REV_FMS_130	Shared Digital Key termination based on expiry date of the Digital Key	FMS-VOT-2
REV_FMS_140	Shared Digital Key termination due to local wipe of device	FMS-RDOT-1
REV_FMS_150	Receiver reports lost/stolen receiver device: suspend keys	R-DOT-4a
REV_FMS_160	Shared key resume from device lost mode in device OEM account	R-DOT-4a
REV_FMS_170	Receiver reports lost/stolen receiver device: remote wipe keys	R-DOT-4b
REV_FMS_180	Security breach on receiver device	R-DOT-5b
Vehicle Unpairing		
REV_FMS_500	Unpairing based on cancellation or expiry date of the digital key service to FMS	FMS-VOT-1
REV_FMS_501	Unpairing by vehicle OEM (vehicle stolen, garage service, ...)	FMS-VOT-3 FMS-VOT-4 FMS-VOT-5
REV_FMS_502	Unpairing by vehicle OEM (FMS security breach)	FMS-VOT-6
REV_FMS_503	Security breach on FMS (FMS not working)	FMS-FMST-1
REV_FMS_504	FMS requests Vehicle unpairing (defleeting)	FMS-FMST-2

13.9 Event notification: SBOD forwarding

For every shared Digital Key related event notification from the vehicle OEM server to the SBOD, the SBOD should notify the FMS. Forwarding in the case of Vehicle OEM SBOD is proprietary to the Vehicle OEM.

Figure 13-22: SBOD eventNotification Forwarding



If required by the vehicle OEM or the fleet management for fraud monitoring, each SBOD or receiver device DK registration request or key termination request processed by the vehicle OEM server and KTS may cause the vehicle OEM server to also send proprietary notifications to inform the FMS of the updated vehicle key information.

13.10 Digital Key Suspension/Termination on Device

13.10.1 REV_FMS_100: Vehicle user terminates Receiver DK in vehicle OEM app

This flow corresponds to REV_210 with vehicle user as “Receiver”.

13.10.2 REV_FMS_110: Vehicle user terminates Shared Digital Key in native app

This flow corresponds to REV_200 with vehicle user as “Receiver”.

13.10.3 REV_FMS_120: FMS terminates Shared Digital Key vehicle user

To terminate a receiver Digital Key, an FMS calls the vehicle OEM server to trigger the termination flow REV/130/140.

13.10.4 REV_FMS_130: Shared key termination based on expiry date of the Digital Key

This flow corresponds to REV_150 with vehicle user as “Friend”.

13.10.5 REV_FMS_140: Shared key termination due to local wipe of device

This flow corresponds to REV_220 with vehicle user as “Receiver”.

13.10.6 REV_FMS_170: Receiver reports lost/stolen receiver device: remote wipe keys

This flow corresponds to REV_170 with vehicle user as “Receiver”.

13.10.7 REV_FMS_180: Security breach on receiver device

This flow corresponds to REV_160 with vehicle user as “Receiver”.

13.10.8 REV_FMS_190: Vehicle user terminates Digital Key in the vehicle

This flow corresponds to REV_100 with vehicle user as “Receiver”.

13.10.9 REV_FMS_150: Receiver reports lost/stolen receiver device: suspend keys

This flow corresponds to REV_300 with vehicle user as “Receiver”.

13.10.10 REV_FMS_160: Shared key resume from device lost mode in device OEM account

This flow corresponds to REV_310 with vehicle user as “Receiver”.

13.11 Vehicle Unpairing

13.11.1 REV_FMS_500: Unpairing based on cancellation or expiry date of the digital key service to FMS

This flow is similar to REV_710.

13.11.2 REV_FMS_501: Unpairing by vehicle OEM (vehicle stolen, garage service,...)

This flow is similar to REV_710 with vehicle OEM server triggering the unpairing process.

13.11.3 REV_FMS_502: Unpairing by vehicle OEM (FMS security breach)

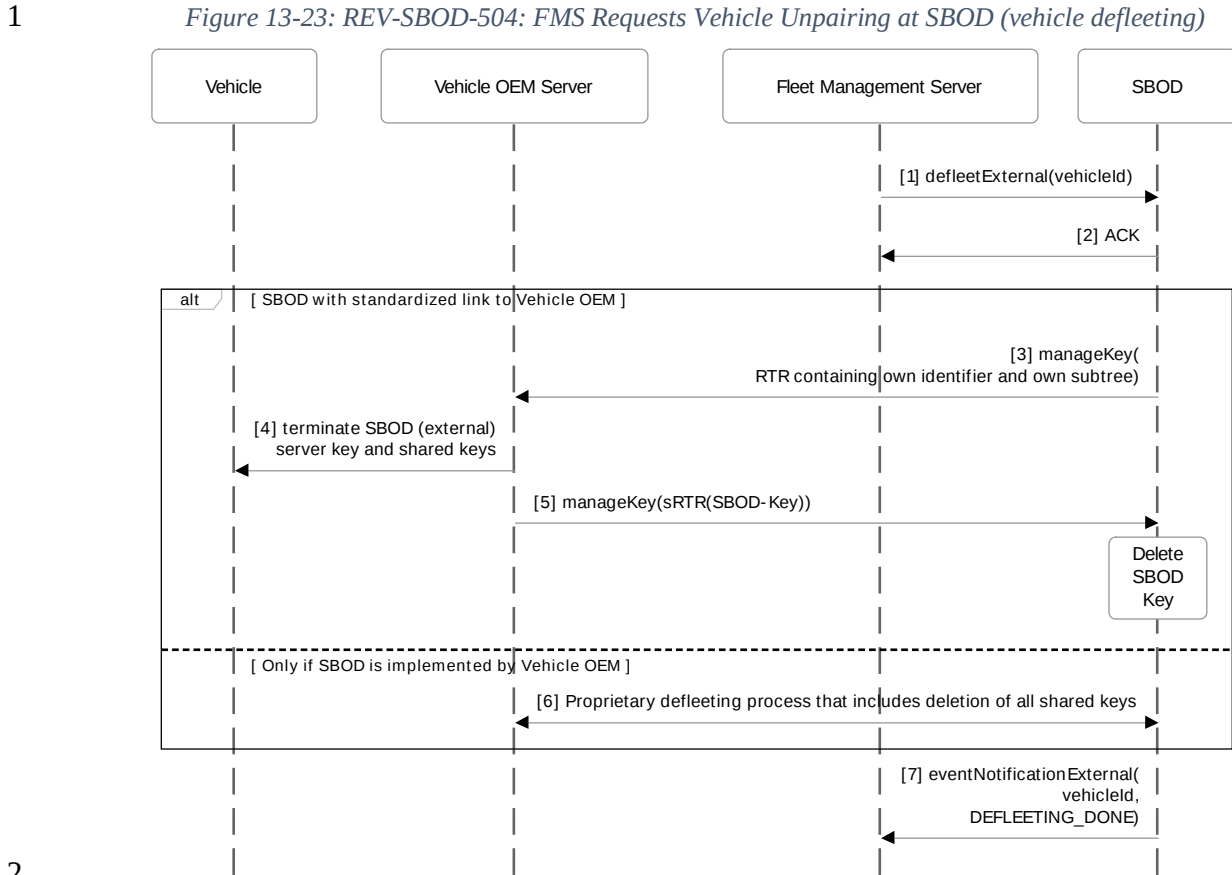
When a security breach of the FMS is detected, the vehicle OEM may unpair the vehicle to remove all (owner and Shared) DKs.

This flow is similar to REV_710 with vehicle OEM server triggering the unpairing process.

13.11.4 REV_FMS_504: FMS requests vehicle unpairing (vehicle de-fleeting)

Defleeting means that the association between the FMS and the vehicle is terminated. This happens when the fleet management provider sells a certain vehicle. Then, for example, the fleet management service needs to be deactivated in the vehicle OEM server and the certain access codes have to get deleted from the vehicle. This process is handled via a proprietary interface between the FMS and the vehicle OEM server and is out of scope of the DK specification.

Defleeting always includes the deletion of all Shared DKs in the receiver devices and in the vehicle.



3 13.11.4.1 Steps 1 and 2 defleetExternal() Request

4 When the vehicle is ready for defleeting, the FMS requests the SBOD to delete the owner key
5 and all shared keys for this vehicle. The SBOD notifies the FMS about the started defleeting
6 process.

7 13.11.4.2 Steps 3 to 5: Vehicle deletes Digital Keys

8 The SBOD sends a remote termination request to the Vehicle OEM server containing its own
9 identifier and own subtree.

10 The vehicle OEM server sends the Digital Key termination request and termination requests for
11 all shared keys to the vehicle via the vehicle OEM telematic link as well as to the KTS.

12 If all steps are successful, the vehicle terminates the SBOD's public key as the owner Digital
13 Key and remaining shared keys.

14 When the vehicle OEM server, KTS, and vehicle have verified termination of the Digital Key
15 and all shared keys, the vehicle OEM server notifies SBOD of successful owner Digital Key
16 termination and deletes its owner key for the requested vehicle.

17

18 When the SBOD is implemented by the Vehicle OEM Steps 3 to 5 can be proprietary.

19 13.11.4.3 Steps 6 to 7 Defleeting Confirmation FMS

20 The SBOD notifies FMS about the successful defleeting process.

1
2
3

14 AUTHENTICATION AND PRIVACY

14.1 Authentication and Privacy Keys

14.1.1 Usage

Sensitive data exchanged between the device and the Vehicle OEM shall be encrypted. The currently identified data includes:

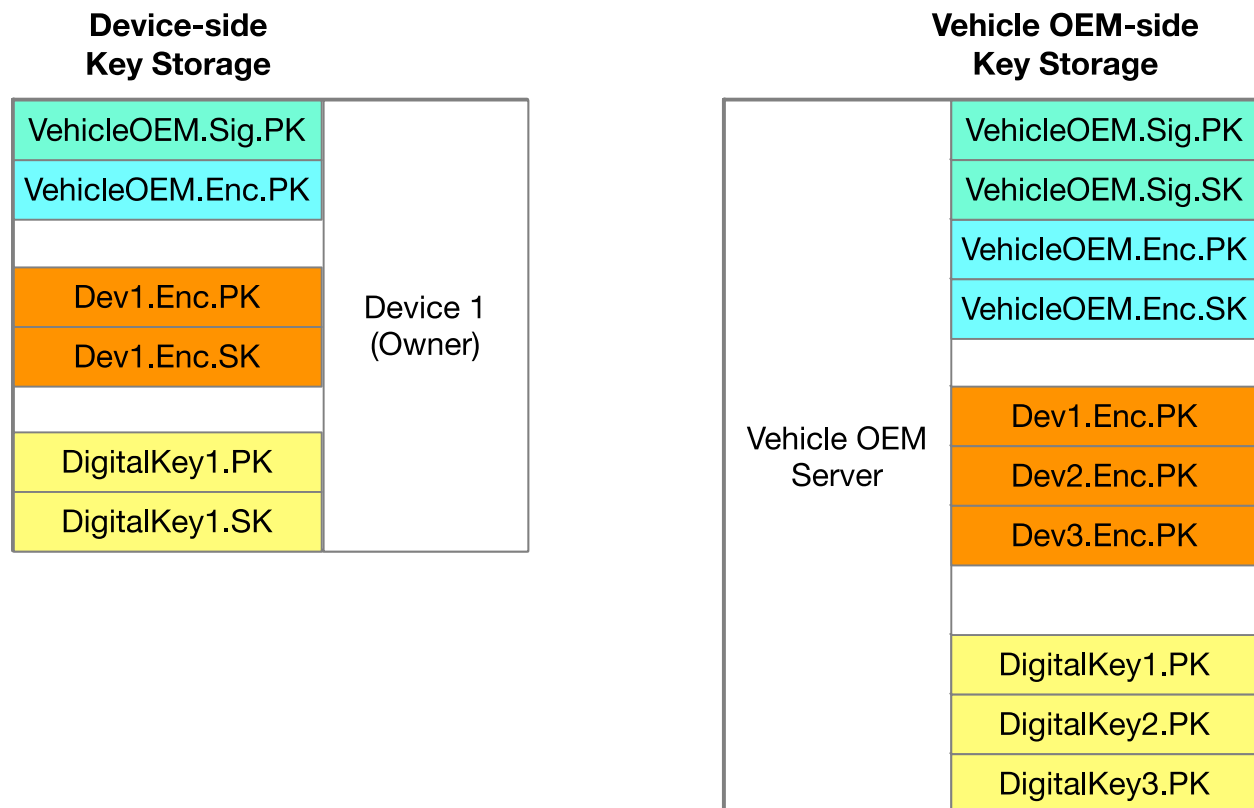
- Key tracking request data (content: attestation package, device encryption public key)
- Notification messages to the sender device that contain shared key information (e.g., the Digital Key identifier of a receiver)
- Remote key deletion requests from the sender device that contain receiver key information (e.g., the Digital Key identifier of a receiver)

Remote key deletion requests that are sent from a device to the Vehicle OEM Server shall be authenticated through a signature.

Key deletion commands that are sent from the Vehicle OEM Server to the target device shall be authenticated through a signature.

[Figure 14-1](#) shows the keys that are used for encryption/decryption and signature verification.

Figure 14-1: Message Authentication and Privacy Encryption



See [Table 14-1](#) for a complete list of involved keys.

1

Table 14-1: Authentication and Privacy Keys

Key	Created by	Purpose	Description
VehicleOEM.Sig.SK	Vehicle OEM Server	Vehicle OEM Server signs the data sent to the device	Safely stored in Vehicle OEM Server
VehicleOEM.Sig.PK	Vehicle OEM Server	Device authenticates the data from the Vehicle OEM Server	Certificate pre-stored in device OS
VehicleOEM.Enc.SK	Vehicle OEM Server	Vehicle OEM Server decrypts the data sent from the device	Safely stored in Vehicle OEM Server
VehicleOEM.Enc.PK	Vehicle OEM Server	Device encrypts the data sent to the Vehicle OEM Server	Certificate pre-stored in device OS
Devx.Enc.SK	Device	Device decrypts the data sent from the Vehicle OEM Server	Created at owner pairing/key sharing, safely stored in the device OS if not already existing
Devx.Enc.PK	Device	Vehicle OEM Server encrypts the data sent to the device	Created at owner pairing/key sharing, sent in key tracking request
DigitalKeyx.SK	Device	Device signs the data sent to the Vehicle OEM Server	This is the Digital Key SK, safely stored in the SE
DigitalKeyx.PK	Device	Vehicle OEM Server authenticates the data sent from the device	This is the Digital Key PK

2 The privacy and authentication keys are used in the following cases, described in detail in
3 Section 17:

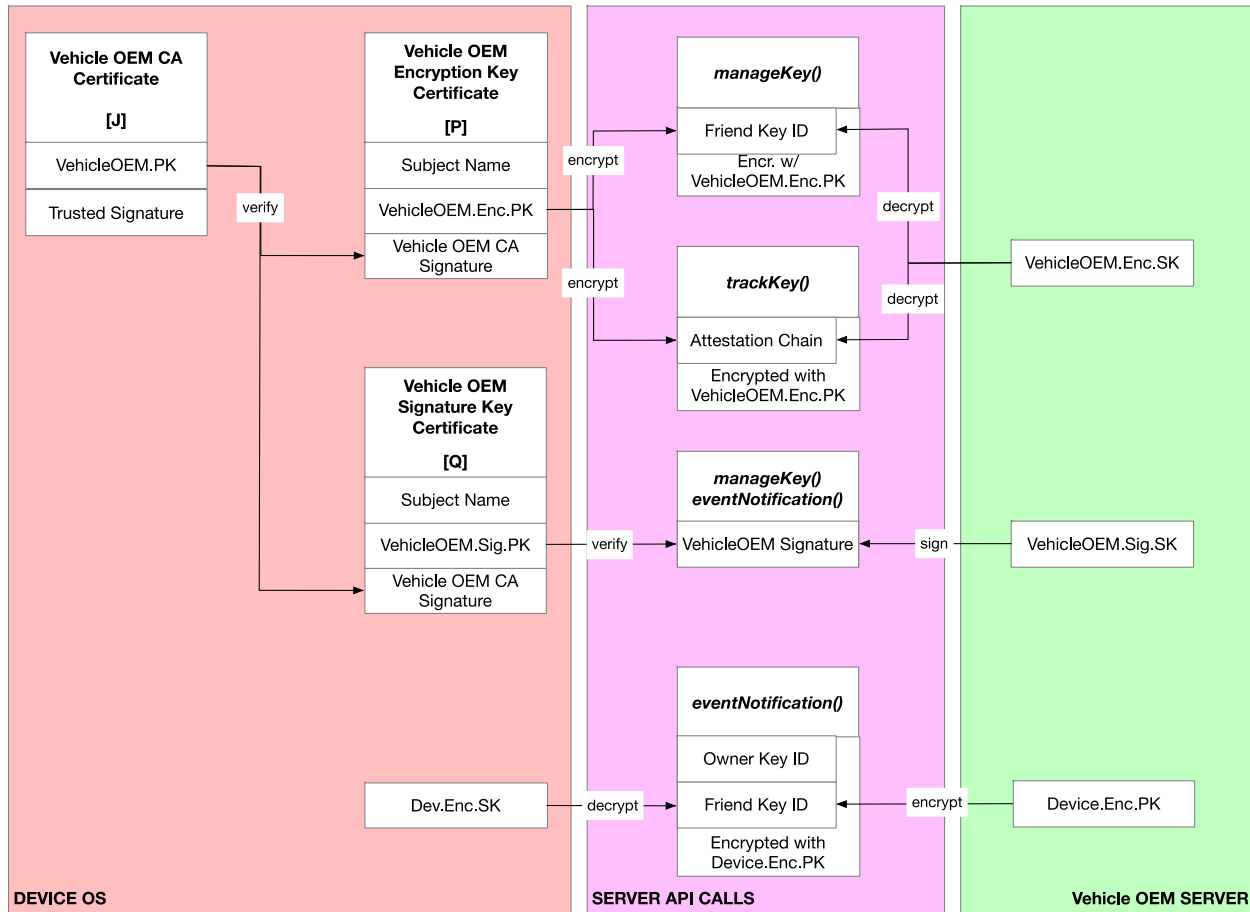
- 4 1. The key tracking request contains Devx.Enc.PK as part of the encrypted payload. The
5 payload is encrypted using VehicleOEM.Enc.PK with the encryption scheme described in
6 Section 14.3. The Vehicle OEM Server stores Devx.Enc.PK associated with the Digital
7 Key identifier provided in the attestation chain.
- 8 2. The remote (Digital Key) deletion request sent from the owner device to the Vehicle
9 OEM Server contains the Digital Key identifiers of owner and shared digital keys in a
10 structure described in Section 14.2. The request data is signed using the owner private
11 key (DigitalKeyx.SK) and encrypted using VehicleOEM.Enc.PK, as it contains the
12 Digital Key identifier of a receiver. The signature scheme is described in Section 14.3.
- 13 3. The key deletion request sent from the Vehicle OEM Server to the target device is signed
14 by the Vehicle OEM Server using the VehicleOEM.Sig.SK. The signature is verified by
15 the target device using VehicleOEM.Sig.PK.
- 16 4. The notification which is sent from the Vehicle OEM Server to the owner device is
17 encrypted using Devx.Enc.PK, as it contains the Digital Key identifier of a receiver.

18 14.1.2 Certificate Chains

19 All public keys used for authentication and privacy are embedded in X.509 certificates except
20 the Device Privacy Encryption Key, and are chained to the trusted root as shown in Figure 14-2.
21 The trust chain of all certificates shall be verified before using them.

The usage of each certificate's public key is described in [Table 14-1](#) and in Section [14.1.1](#). All private keys are assumed to be stored safely where they have been created.

Figure 14-2: Authentication and Privacy Encryption Certificate Chain



14.1.2.1 [H] – Digital Key Creation Attestation

See Section [16.2.8](#) for a description of this certificate.

14.1.2.2 [J] – Vehicle OEM CA Certificate

See Section [16.2.9](#) for a description of this certificate.

14.1.2.3 [P] – Vehicle OEM Privacy Encryption Certificate

This certificate is generated by the Vehicle OEM Server and provided to the Device OEM. Each Device OEM shall receive a different public key in the certificate. The transfer of the certificate from the Vehicle OEM Server via the Device OEM Server to the device is out of scope of this specification. The certificate should have a lifetime defined by Vehicle OEM policy. Revocation may be achieved by stopping the renewal process or through revocation methods (CRL, OCSP), which are out of scope of this specification.

The certificate shall have proprietary extension using the assigned OID listed in Appendix [B.2](#). The extension is defined as shown in [Listing 14-1](#) and [Listing 14-2](#).

Listing 14-1: Vehicle OEM Privacy Encryption Certificate Extension Schema

```
VehicleOEMPrivacyEncryptionCertificateExtensionSchema ::= SEQUENCE
{
    extension_version  INTEGER (1 ..255)
}
```

Listing 14-2: Vehicle OEM Privacy Encryption Certificate Extension Data

```
vehicle-oem-privacy-encryption-cert-extension-data VehicleOEMPrivacyEncryptionCertificateExtensionSchema ::=
{
    extension_version  1  --value shall be 1
}
```

14.1.2.4 VehicleOEM Signature Key Certificate

This certificate is generated by the Vehicle OEM Server and provided to the Device OEM. Each Device OEM shall receive a different public key in the certificate. The transfer of the certificate from the Vehicle OEM Server via the Device OEM Server to the device is out of scope of this specification. The certificate should have a lifetime defined by Vehicle OEM policy. Revocation may be achieved by stopping the renewal process or through revocation methods (CRL, OCSP), which are out of scope of this specification

The certificate shall have a proprietary extension using the assigned OID listed in Appendix B.2.2. The extension is defined as shown in [Listing 14-3](#) and [Listing 14-4](#).

Listing 14-3: Vehicle OEM Signature Certificate Extension Schema

```
VehicleOEMSignatureCertificateExtensionSchema ::= SEQUENCE
{
    extension_version  INTEGER (1 ..255)
}
```

Listing 14-4: Vehicle OEM Signature Certificate Extension Data

```
vehicle-oem-signature-cert-extension-data VehicleOEMSignatureCertificateExtensionSchema ::=
{
    extension_version  1  --value shall be 1
}
```

14.1.2.5 Device Privacy Encryption Key

This key is generated by the device and provided to the Vehicle OEM Server in the TrackKey() API call (see Section [17.7.3](#)). Each device shall generate a unique device encryption key pair per Digital Key [H].

14.2 Remote Termination Requests

The Remote Termination Request shall be sent when either the terminator device or the Vehicle OEM Server requests the deletion of a Digital Key on a target device.

If the terminator device originates the Remote Termination Request, the request structure in [Table 14-3](#) shall be used.

If the Vehicle OEM Server or the vehicle itself originate the Remote Termination Request, the request structure in [Table 14-4](#) shall be used. A specifically defined source key id indicates whether the request is originated by the vehicle or the Vehicle OEM Server (user account, subscription ended, etc.).

When the key id of the target key is not known to the source device, the target key id tag shall be absent.

If the key id of the target key is known to the source device but not to the server and vehicle, the slot identifier tag shall be used to identify the key to terminate. The corresponding slot identifier shall be deleted in the server and vehicle. When the target device attempts to track the key after its deletion, the track key request shall be rejected.

In some cases, the slot identifier might not be known. In those cases, the key identifier shall be used to identify the key to be deleted.

[Table 14-2](#) describes the data fields of the Remote Termination Request.

Table 14-2: Remote Termination Request Data Fields

Tag			Length (bytes)	Description	Field is	Domain Version
5F21 _h			20	subject key identifier (key id) of the source Digital Key	mandatory	D-VS
7F23 _h			variable	list of key identifier and slot identifier pairs of the target Digital Key(s)	Conditional: always present for sRTR. For dRTR, only present if 7F2F _h is absent.	D-VS
	61 _h		variable	pair of key identifier and slot identifier of 1 st Digital Key	mandatory	D-VS
		50 _h	20	subject key identifier	mandatory for sRTR, conditional for dRTR	D-VS
		57 _h	variable	slot identifier	optional for sRTR, conditional for dRTR	D-VS
	61 _h		variable	pair of key identifier and slot identifier of 2 nd Digital Key	conditional	
		50 _h	20	subject key identifier	mandatory for sRTR, conditional for dRTR	
		57 _h	variable	slot identifier	optional for sRTR, conditional for dRTR	
7F2F _h			variable	(List of) groupIdentifiers of accounts to terminate (terminate all keys on that device account for the vehicle determined by the source DK)	Conditional: never present for	D-VS

Tag			Length (bytes)	Description	Field is	Domain Version
					sRTR. For dRTR only present if 7F23 _h is absent.	
	61 _h		variable	Pair of groupIdentifier and deletion parameters	conditional	D-VS
		4F _h	2	GroupIdentifier of account 1 to terminate	conditional	D-VS
		47 _h	1	0: delete account only, 1: delete entire subtree	conditional	D-VS
...						

The sRTR shall contain tag 7F23_h and the dRTR shall contain either tag 7F23_h (for implementations that don't manage keys on account level) or tag 7F2F_h. Requests containing both tags shall not be accepted and trigger 50126 sub-status codes indicating invalid remote termination request, as per manageKey (see [17.8.2](#)).

A request sent from the sender device to the vehicle OEM server (dRTR) may contain one or more pairs of target slot identifiers and target key identifiers. The maximum number of keys in the request may be defined by the vehicle OEM, otherwise it is limited to 16.

The dRTR can contain one or more groupIdentifiers. The number is limited to 32. For every account (i.e., groupIdentifier) it is possible to specify whether only the account shall be deleted or the full sharing accounts subtree.

If one account is deleted, then the subtree shall be re-attached by using the sender groupIdentifier of the deleted accounts.

Termination requests from devices shall only be accepted by the vehicle OEM server if the terminator device has the appropriate management capabilities as defined by AccountRole. Otherwise, the vehicle OEM server responds with 50127 sub-status code indicating insufficient management rights, as per manageKey (see [17.8.2](#)). For tag 7F23_h, the request shall contain either subject key identifier or slot identifier in element (tag 61_h). It should contain both if they are available.

The input data fields for signature creation for the remote termination request for the terminator device and the Vehicle OEM Server are described in [Table 14-3](#) and [Table 14-4](#), respectively.

The signature creation for the remote termination request is described in [Table 14-3](#).

Table 14-3: Remote Termination Request from Terminator Device

Tag	Length (bytes)	Description	Field is	Domain Version
7F40 _h	variable	Remote Termination Request from Terminator device	mandatory	D-VS
content of Table 14-2			mandatory	D-VS
SIG-DAT: content of Table 15-58 (Signature Data fields) with: arbitrary_data_ = SHA-256 hash value of Table 14-2			mandatory	V-OD-FW

Tag	Length (bytes)	Description	Field is	Domain Version
9E _h	64	signature with the private key of the sender Digital Key over fields from SIG-DAT	mandatory	V-OD-FW

The signer of the request is determined by the key id present in the manageKey call as per table Table 13-5.

Table 14-4: Remote Termination Request from Vehicle OEM Server

Tag	Length (bytes)	Description	Field is	Domain Version
7F39 _h	variable	Remote Termination Request from Vehicle OEM Server	mandatory	D-VS
		content of Table 14-2 with: source key id = <vehicle key id>, if originated in vehicle source key id = <vehicle OEM key id>, if originated in Vehicle OEM Server source key id = <terminator key id>, if originated by terminator device via Vehicle OEM Server		D-VS
5F22 _h	20	Subject key identifier of the certificate to be used for signature verification (VehicleOEM.Sig.Cert)	Conditional. Present if more than one signature verification certificates have been issued by the Vehicle OEM	D-VS
9E _h	64	signature with the private key of the Vehicle OEM Server (VehicleOEM.Sig.SK) over fields from Table 14-2	mandatory	D-VS

The <vehicle OEM key id> is defined as 20 bytes of FF_h. It is used when the key was deleted by the Vehicle OEM (e.g., subscription ended).

The <vehicle key id> is defined as 20 bytes of EE_h. It is used when the key was deleted in the vehicle.

<terminator key id> is provided in the remote termination request coming from the terminator. It is used when the key was deleted by the terminator device.

The signature covers tags 5F21_h and 7F23_h, inclusive of tags and lengths, as defined in [Table 14-2](#).

When receiving a remote termination request from the terminator device ([Table 14-3](#)), the Vehicle OEM Server shall verify the following before executing the termination request:

- Value of the hash of [Table 14-2](#) matches the arbitrary_data value provided in SIG-DAT
- Signature is correct
- Signer key of dRTR matches with the subject key identifier in the dRTR ([Table 14-3](#)).
- Shared key AccountRoles do not upgrade shared keys beyond what is allowed per AccountRole (e.g., allow shared keys to share while sender can only share keys that cannot share anymore).

- Terminated key is not in the terminator key subtree if the terminator key has only subtree deletion rights according to AccountRoles.

When receiving a remote termination request from the Vehicle OEM Server ([Table 14-4](#)), the receiver device shall verify the following before executing the termination request:

- Source key ID corresponds to the key ID of the originator
- Signature is correct

The Device OEM Server shall ensure the following security properties:

- Each termination request from a mobile device to the Device OEM Server shall contain a signed terminationAttestation (device deletes its own key) or remoteTerminationRequest (terminator deletes shared key).
- Each termination request and each termination response from a mobile device to the Device OEM Server shall contain the Vehicle OEM proprietary data subsection of the private mailbox.
- The user can delete only their own keys in the device portal, and not the shared keys.
- When keys are deleted in the Device OEM portal (see [Table 13-4](#)) and the device could not be reached, the Device OEM Server is not required to provide a remoteTerminationRequest to the Vehicle OEM Server.

14.2.1 Example of RTR Signature Verification

RTR –

```
7F397C5F2114FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF7F2320611E5014444DF
CECAA57A75B4C42D60A7EB1868B793BEDE05706000F000000019E40CD846C5BCFAB51
41CE129A7C197129220EB5747B9A80C1726DEA200FFA42B2178235452B60DD60AA8D7F
9EB43EA486A87C91B9BC9886D83DAE9583F1112D183A
```

SIG –

```
CD846C5BCFAB5141CE129A7C197129220EB5747B9A80C1726DEA200FFA42B217823545
2B60DD60AA8D7F9EB43EA486A87C91B9BC9886D83DAE9583F1112D183A
```

TBS –

```
5F2114FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF7F2320611E5014444DFCECAA
57A75B4C42D60A7EB1868B793BEDE05706000F00000001
```

PK -

```
043517d1a4043c6738a2fc0458fec4ae94c833b7fac59bf697ee65979bcd64cbc3ce7323a6c5f0c6
6e9500734e33f0bd93c280b7299fc85674bbc64766492e25
```

```
>>> pubkey = bytes.from-
```

```
hex('3517d1a4043c6738a2fc0458fec4ae94c833b7fac59bf697ee65979bcd64cbc3ce7323a6c5f0
c66e9500734e33f0bd93c280b7299fc85674bbc64766492e25')
```

```
>>> data = bytes.from-
```

```
hex('5F2114FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF7F2320611E5014444DFCE
CAA57A75B4C42D60A7EB1868B793BEDE05706000F00000001')
```

```

1 >>> sig = bytes.from-
2 hex('CD846C5BCFAB5141CE129A7C197129220EB5747B9A80C1726DEA200FFA42B2178
3 235452B60DD60AA8D7F9EB43EA486A87C91B9BC9886D83DAE9583F1112D183A')
4 >>> vk = ecdsa.VerifyingKey.from_string(pubkey, curve = curve)
5 >>> vk.verify(sig, data, hashfunc = hashfunc)
6 True

```

14.3 Encryption and Signature Verification Schemes

The data to be privacy-encrypted is described in Section 17. The encryption scheme to be used is described in Section 17.12.2. Signatures of the commands shall use ANSI X9.62 ECDSA algorithm with SHA-256 (see [23] and [24]).

14.4 OEM App Data Attestation

OEM App Data Attestation shall be used to obtain a signature over data provided by the OEM App, using the endpoint.SK. In response, all fields of the returned attestation shall be provided to the Vehicle OEM App. For OEM App Data Attestation, a request for User Consent shall be enforced. Enforcement of User Authentication is optional.

Table 14-5 describes the input fields of the OEM App Data Attestation.

Table 14-5: OEM App Data Attestation Input

ASN.1 Tag	Length (bytes)	Description	Field is
04 _h	variable	App Bundle Identifier	mandatory
04 _h	variable	OEM App Data	mandatory
04 _h	variable	Nonce, defined by framework	mandatory

Attestation is generated as described in Listing 14-5.

Listing 14-5: OEM App Data Attestation Processing

```

Input: Table 14-5
output: attestation
begin
  perform user consent
  return error if user did not consent
  perform user authentication if required
  arbitrary_data = SHA-256 hash value of Table 14-5
  send SIGN command to retrieve attestation as described in Table 15-65
  return attestation
end

```

15 DIGITAL KEY APPLET

15.1 Introduction

The Digital Key applet is aimed at providing multi-purpose SE-based transaction mechanisms combined with peer-to-peer key distribution and a data storage system with strong security and privacy properties. Three types of contactless transaction may be used: standard transaction (see Section 7), fast transaction (see Section 8), and check presence transaction (see Section 10).

In this specification, two applet implementation models are provided, depending on the Device OEM's implementation or the Digital Key deployment model.

- **SE-centric applet model:** For this model, the Device OEM CA Certificate's corresponding public key is protected by the SE, and the non-SE endpoints (e.g., vehicle, server, etc.) are verified by the SE.
- **Framework-centric applet model:** For this model, the Device OEM CA Certificate's corresponding public key is protected by the device OS native key store, and the non-SE endpoints (e.g., vehicle, server, etc.) are verified by the framework.

15.2 Keys and Data

This section defines some of the key and data elements used later in Section 15 to help understand the flow diagrams.

Cryptogram: MAC value calculated over unique public transaction data. The cryptogram proves that the device is in possession of the same symmetric key as the vehicle.

Kpersistent: Symmetric long-term key that is used to derive encryption and MAC session keys. It is stored in NVM on both vehicle and device sides.

Kenc: Derived symmetric key used to encrypt confidential commands and responses payloads

Kmac: Derived symmetric key used to calculate command MACs

Krmac: Derived symmetric key used to calculate response MACs

Kcmac: Derived symmetric key used to calculate cryptograms

Kseed: Symmetric that is used to derive keys for confidential data encryption and MAC

Keenc: Derived symmetric key used to encrypt confidential data

Kemac: Derived symmetric key used to compute MAC on encrypted confidential data

Kdh: Symmetric key generated by a Diffie-Hellman operation

***ePK/*eSK:** denotes ephemeral public and private keys

***PK/*SK:** denotes public and private keys

transaction_identifier: randomly generated nonce value and used on vehicle and device sides for symmetric key derivation and MAC/signature generation/verification

vehicle_identifier: unique identifier of the vehicle. On device side, it is used to look up the correct endpoint

15.3 Applet Implementation

15.3.1 Introduction

The following subsections describe internal data structures and APDU commands for the applet. This specification supports only one Digital Key applet protocol version. The Digital Key applet protocol version is set to 0100_h.

15.3.1.1 TLV Field

The various Tag Length Value fields presented in this document shall comply with the BER-TLV format as defined in ISO 7816-4 [1]. The TLV fields shall be ordered as described in this specification; a different field order is considered invalid unless specified otherwise. The nesting level is represented by indentation of Tag values in the Tag column.

15.3.1.2 Applet Memory Types

The naming conventions for the memory types used in the applet are as follows:

- Variable prefixed with “nvm.” Indicates the memory used is persistent after power off.
- Variable prefixed with “cod.” indicates the memory used is volatile and erased after applet deselection (caused by contactless field off or explicit deselection).
- Variable noted as (nvm/cod).object.variable indicates the variable is an object member.
- Variable with neither memory prefix nor object prefix indicates a local volatile variable erased after exiting the current programming context.
- All variables and objects are globally available at the instance level except the local variables defined above.
- Variable suffixed with [] indicates a table of variables.
- Variable suffixed with * indicates all the variables names starting with the string present before *.

15.3.1.3 Endpoint Life Cycle States

state_endpoint_active

In this state, the endpoint accepts all commands allowed by its configuration.

State_endpoint_terminated

In this state, the endpoint only accepts the TERMINATE ENDPOINT and DELETE ENDPOINT commands.

15.3.1.4 Communication Interfaces

The wired interface connects the SE to the device’s Digital Key framework through physical wires inside the device. The contactless interface connects the SE to the NFC reader in the vehicle through radio interface. The notifications sent from the applet to the Digital Key framework (denoted as notify_*) may be implemented as described in Section 15.3.1.9 or using a proprietary method.

1

Table 15-1: Command Availability on Interfaces

Command	Wired Interface	Contactless Interface	Comment	Section
SELECT	Yes	Yes		15.3.2.1
CREATE CA	Yes	No		15.3.2.2
DELETE CA	Yes	No		15.3.2.3
CREATE ENDPOINT	Yes	No		15.3.2.4
TERMINATE ENDPOINT	Yes	No		15.3.2.5
DELETE ENDPOINT	Yes	No		15.3.2.6
GET PRIVATE DATA	Yes	No		15.3.2.18
SET PRIVATE DATA	Yes	No		15.3.2.19
AUTHORIZE ENDPOINT	Conditional	No	Availability configured in Section 15.3.2.4	15.3.2.7
CONVERT ENDPOINT	Yes	No		15.3.2.29
SET CONFIDENTIAL DATA	Yes	No		15.3.2.20
CREATE ENCRYPTION KEY	Yes	No		15.3.2.17
VIEW	Yes	No		15.3.2.8
SETUP ENDPOINT	Yes	No		15.3.2.21
SETUP INSTANCE	Yes	No		15.3.2.22
AUTH0	Conditional	Mandatory	Availability configured in 15.3.2.4	15.3.2.9
AUTH1	Conditional	Mandatory	Availability configured in Section 15.3.2.4	15.3.2.10
PRESENCE0	No	Yes		15.3.2.11
PRESENCE1	No	Yes		15.3.2.12
READ BUFFER	Yes	No		15.3.2.13
WRITE BUFFER	Yes	No		15.3.2.14
EXCHANGE	Conditional	Yes	Availability configured in Section 15.3.2.4	15.3.2.15
CONTROL FLOW	Yes	Yes		15.3.2.16
SIGN	Conditional	No	Availability configured in Section 15.3.2.4	15.3.2.23
MANAGE UA	Conditional	No	Availability configured in Table 15-5	15.3.2.24
CREATE RANGING KEY	Conditional	No	Availability depends on platform	15.3.2.26
DELETE RANGING KEYS	Conditional	No	Availability depends on platform	15.3.2.27
GET NOTIFICATION	Optional	No	Availability depends on the implementation	15.3.2.28

Implementations may require the use of a SCP such as SCP03 [7] or SCP11 [28] between the Digital Key applet and the Digital Key framework. In these cases, the Digital Key applet shall support usage of a SCP for the following commands: CREATE ENDPOINT, TERMINATE ENDPOINT, DELETE ENDPOINT, GET PRIVATE DATA, SET PRIVATE DATA, VIEW, SETUP ENDPOINT, SETUP INSTANCE, READ BUFFER, WRITE BUFFER, SIGN, AUTHORIZED ENDPOINT, SET CONFIDENTIAL DATA, CREATE ENCRYPTION KEY, MANAGE UA, DELETE RANGING KEYS and CONVERT ENDPOINT. This option of the Digital Key applet implementation is called Option A (i.e., DK applet that supports SCP) in this section. The provisioning of secure channel keys is out of scope of this specification. In case the response data does not fit in the response APDU due to secure channel overhead (e.g., due to the addition of padding or response_mac), the applet shall respond with an error status word.

The commands set over the contactless interface are standardized commands. SELECT, AUTH0, AUTH1, PRESENCE0, PRESENCE1, EXCHANGE, and CONTROL FLOW commands are mandatory to be supported; all other commands are optional.

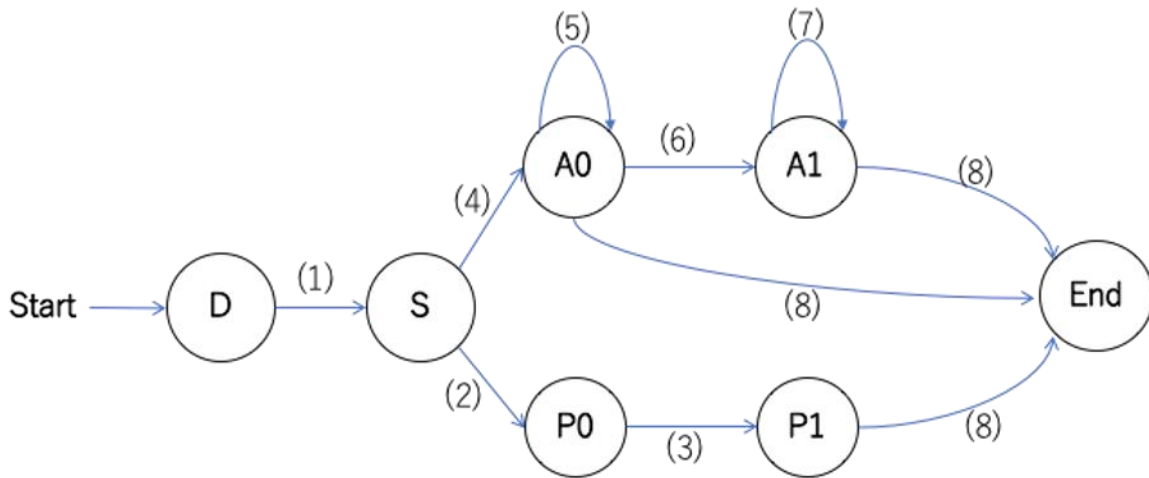
Over the wired interface, implementations may require the support of supplementary logical channels in addition to support for the basic logical channel. This option of the Digital Key applet implementation is called Option B (i.e., DK applet that supports supplementary logical channels) in this section.

15.3.1.5 Command Flows

Figure 15-1 describes the allowed command flows over the contactless interface; other command flows are rejected. The CONTROL FLOW command (which can be used anywhere after the SELECT command) is not represented in this figure.

1

Figure 15-1: Authentication Command Flows Over Contactless Interface



State

D: Applet Deselected
S: Applet Selected
A0: AUTH0 done
A1: AUTH1 done
P0: PRESENCE0 done
P1: PRESENCE1 done

Action

(1) SELECT
(2) PRESENCE0
(3) PRESENCE1
(4) AUTH0
(5) EXCHANGE
(6) AUTH1
(7) EXCHANGE
(8) NFC Link Teardown or
NFC Reset Procedures

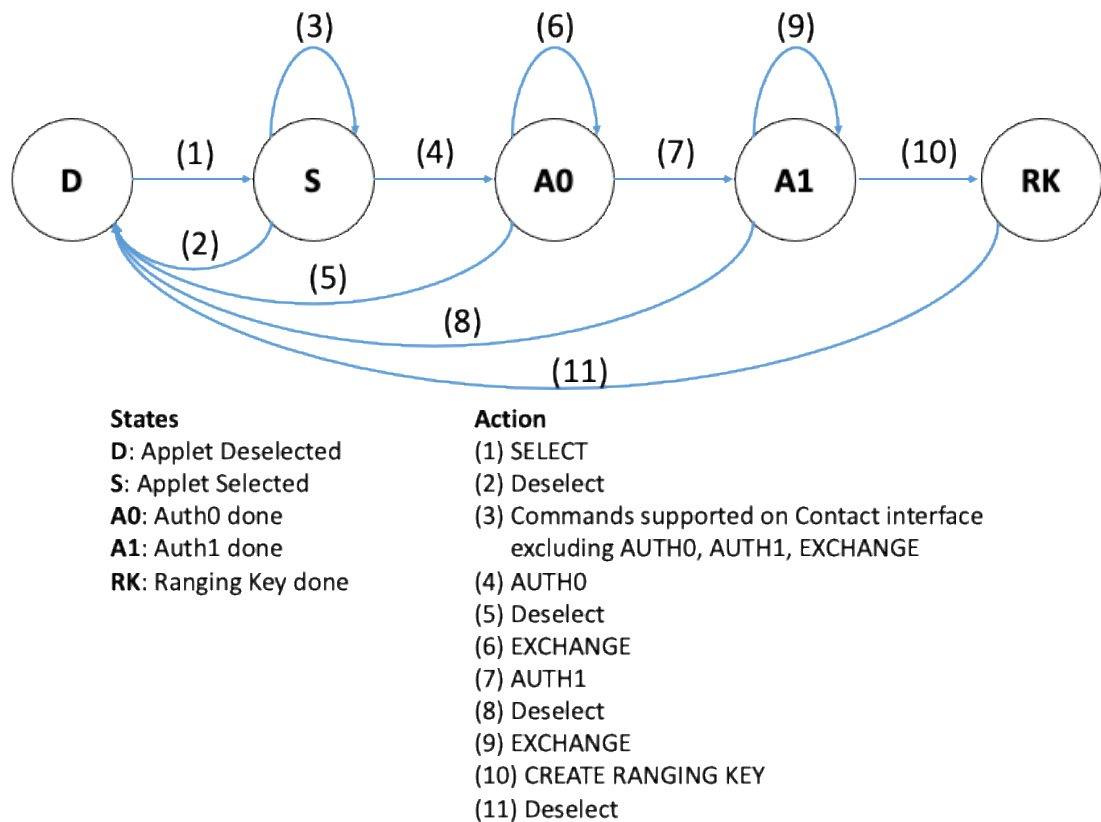
2

3 Figure 15-2 describes the allowed command flows over the wired interface. The CONTROL
4 FLOW command (which can be used anywhere after the SELECT command) is not represented
5 in this figure.

6

Figure 15-2: Authentication Command Flows Over Wired Interface

7



In state A0, only AUTH1 and EXCHANGE commands are accepted. In state A1, only EXCHANGE and CREATE RANGING KEY commands is accepted.

GET NOTIFICATION command, if implemented, may be sent at any time after the execution of a command. It has not impact on the state

1

2 15.3.1.6 Generic Error Handling

3 This section applies to all listed commands. It describes the generic status words to be retrieved
4 in case of error during basic input command checking.

5 The basic input command checking includes checking that an INS is allowed on a given
6 interface, that the CLA is consistent, that P1/P2 bytes have valid values, that Lc is in valid range,
7 and checking the format of the payload.

8 The basic input command checking is executed before the steps described in the Listings below
9 and can result in the error status words described in [Table 15-2](#).

10 This specification does not describe all other possible errors that can occur during a command
11 processing when not specified in the command listing. In such cases, the choice of the status
12 word is left to the implementer. However, usage of status word 6400_h is highly recommended to
13 ensure interoperability.

14 Digital Key applet should return an error if an APDU command includes unexpected or unknown
15 parameters or tags

1

Table 15-2: Generic Status Words

Status Word	Comment
6400 _h	No specific diagnostic
6A80 _h	Wrong payload format
6A84 _h	Not enough memory
6A86 _h	Wrong P1 or P2
6A88 _h	Reference data not found
6700 _h	Wrong command payload length
6D00 _h	Wrong INS code
6E00 _h	Wrong CLA code
9000 _h	Command successfully executed

2 15.3.1.7 Class byte coding

3 The class byte coding of all commands shall conform to Section 5.4 of [1]. The value of the class
4 byte depends on which logical channel (basic or supplementary) the command is addressed for
5 and on whether or not secure messaging is used. As a result:

- 6 • When the implementation supports Option B, a range of class byte values is applicable to all
7 the commands available on the wired interface. For commands addressed to the basic logical
8 channel or supplementary logical channels 1, 2, and 3, the class byte coding is defined in
9 Table 2 of [1]. For commands addressed to supplementary logical channels 4 through 19, the
10 class byte coding is defined in Table 3 of [1].
- 11 • When the implementation supports Option A, a range of class byte values is applicable to the
12 subset of commands defined in Section 15.3.1.4 when a secure channel is used between the
13 Digital Key applet and the Digital Key framework.

14 For commands addressed to the basic logical channel or to supplementary logical channels 1, 2,
15 and 3:

- 16 • Bit b8 is set to 0 for the SELECT command and to 1 for all other commands defined in this
17 specification.
- 18 • Bits b7 to b5 are set to 0.
- 19 • When a secure channel is used, bits b4 and b3 are set to 01; otherwise they are set to 00.
- 20 • Bits b2 and b1 indicate the logical channel number from 0 to 3.

21 For commands addressed to the supplementary logical channel 4 through 19:

- 22 • Bit b8 is set to 0 for the SELECT command and to 1 for all other commands defined in this
23 specification.
- 24 • Bit b7 is set to 1.
- 25 • When a secure channel is used, bit b6 is set to 1; otherwise it is set to 0.
- 26 • Bit b5 is set to 0.
- 27 • Bits b4 to b1 indicate the logical channel number from 4 through 19 (i.e., 0000 for logical
28 channel 4 through 1111 for logical channel 19)

[Table 15-3](#) lists the four different ranges of class byte values to be supported for the different commands, depending on the options implemented by the Digital Key applet. The determination of whether a command shall use CLA1, CLA2, CLA3, or CLA4 range is defined in the command description provided in Section [15.3.2](#).

Table 15-3: Coding of the Different Ranges of Class Byte Values

Range of values of the class byte (CLA)	Over the contactless interface (hex)	Over the wired interface (hex)			
		DK applet implementation does not support Options A or B.	DK applet implementation supports Option A but does not support Option B.	DK applet implementation supports Option B but does not support Option A.	DK applet implementation supports Options A and B.
CLA1	00	00	00	00 – 03 or 40 – 4F	00 – 03 or 40 – 4F
CLA2	N/A	80	80 or 84	80 – 83 or C0 – CF	80 – 87 or C0 – CF or E0 – EF
CLA3	80	80	80 or 84	80 – 83 or C0 – CF	80 – 83 or C0 – CF
CLA4	84	84	84	84 – 87 or E0 – EF	84 – 87 or E0 – EF

15.3.1.8 INSTALL for INSTALL parameters

This command defined in GlobalPlatform specification [20] creates a Digital Key applet instance and allocates the needed memory.

The following values are provided in tag C9_h during the INSTALL for INSTALL command. Refer to GlobalPlatform specification [20] for formatting of this command.

The INSTALL for INSTALL command might include additional parameters, depending on the Device OEM policy for managing the non-volatile memory available for the Digital Key applet (e.g., using the Non-volatile Memory Quota parameters defined in GlobalPlatform [\[20\]](#)).

Table 15-4: INSTALL for INSTALL Content of Tag C9

Tag	Length (bytes)	Description	Field is
A0 _h	1	install_param_endpoint_count, in range 1–255	mandatory
A1 _h	1	install_param_instance_CA_count, in range 1–255	mandatory
A2 _h	2	install_param_internal_buffer_size (MSB LSB), in range 1024–max (where, “max” is defined by the Device OEM)	mandatory
A3 _h	2	install_param_max_allocatable_private_mailbox_size (MSB LSB), in range 0–2048	mandatory

Tag	Length (bytes)	Description	Field is
A4 _h	2	install_param_max_allocatable_confidential_mailbox_size (MSB LSB), in range 0–2048	mandatory
A6 _h	variable	install_param_oid_external_ca	mandatory
A7 _h	variable	install_param_oid_instance_ca	mandatory
A8 _h	variable	install_param_oid_endpoint	mandatory
63 _h	1	applet_implementation_options	optional

1 Install_param_endpoint_count

2 The maximum number of endpoints that can be created on an instance of the applet.

3 Install_param_instance_CA_count

4 The maximum number of Instance CA that can be created on an instance of the applet.

5 Install_param_internal_buffer_size

6 The size in bytes of the internal buffer described in Section [15.3.2.13](#). This value is set by the
7 Device OEM according to Device OEM service policy, which is out of scope of this
8 specification.

9 Install_param_max_allocatable_private_mailbox_size

10 The maximum size for private mailbox per endpoint. This value is set by the Device OEM
11 according to its service policy, which is out of scope of this specification.

12 Install_param_max_allocatable_confidential_mailbox_size

13 The maximum size for confidential mailbox per endpoint. This value is set by the Device OEM
14 according to its service policy, which is out of scope of this specification.

15 Applet_implementation_options

16 The applet_implementation_options is encoded as defined in Table 15-5.

17 If this field is not present, the applet implementation model is the framework-centric applet
18 model, and the SE root of trust certificate chain model is Variant 1, as depicted in [Figure 16-1](#),
19 and the MANAGE UA command is not implemented or disabled.

20 *Note:* Tag A6_h, A7_h and A8_h have fixed values as defined in Appendix [B.2](#), however they are
21 included here to allow for potential evolutions in the future.

Table 15-5: Applet Implementation Options

b8	b7	b6	b5	b4	b3	b2	b1	Description
-	-	-	-	-	-	-	0	applet_implementation_model is framework-centric applet model
-	-	-	-	-	-	-	1	applet_implementation_model is SE-centric applet model
-	-	-	-	-	-	0	-	SE Root of Trust certificate chain models is Variant 1
-	-	-	-	-	-	1	-	SE Root of Trust certificate chain models is Variant 2
-	-	-	-	-	0	-	-	MANAGE UA command is not available
-	-	-	-	-	1	-	-	MANAGE UA command is available

b8	b7	b6	b5	b4	b3	b2	b1	Description
0	0	0	0	0	-	-	-	RFU (0)

Listing 15-1: INSTALL for INSTALL Processing

```
1  input : install_param *
2  output : none
3  begin
4    check install_parameters
5    atomic start
6      allocate cod.internal_buffer using the size defined by install_param_internal_buffer_size
7      If not enough memory
8        return 6A84h
9      reserve space using the number of endpoints defined by
10     install_param_endpoint_count,
11     install_param_instance_CA_count
12     create dummy_endpoint
13     generate key pair endpoint_PK, endpoint_SK according to Listing 15-41
14     nvmmwrite nvm. Dummy_endpoint.SK ← endpoint_SK
15     nvmmwrite nvm. Dummy_endpoint . Kpersistent , fill with random
16     nvmmwrite nvm. Dummy_endpoint . kpersistent_established ← true
17     nvmmwrite nvm.dummy_endpoint.terminated ← false
18     nvmmwrite nvm.dummy_endpoint.option_group_1 ← 0Fh (b4–b7 are set to zero)
19     nvmmwrite nvm.dummy_endpoint.option_group_2 ← 00h (b0,b1,b7 are set to zero)
20     nvmmwrite nvm.supported_protocol_versions_tlv ← TLV as per Table 15-12
21   atomic commit
22 end
```

The following values define the Protocol Data Type A (Card Emulation Mode) tag (86_h) and the Protocol Data Type B (Card Emulation Mode) tag (87_h), which are provided in tag A0_h (Contactless Protocol Parameters tag) of tag EF_h (System Specific Parameters tag) during the INSTALL for INSTALL command. Refer to GlobalPlatform Card Specification Amendment C [20] for formatting of these parameters.

Table 15-6: Values for Protocol Data Type A (tag ‘86’)

Tag	Length (bytes)	Value Description	Value in Protocol Parameter Data (Tag ‘A0’)	Value in Protocol Parameter Mandatory Mask (Tag ‘A1’)
80 _h	1	Unique Identifier LV structure	00 _h	00 _h
81 _h	1	SAK	20 _h	24 _h
82 _h	2	ATQA	0400 _h	39F0 _h
83 _h	1	ATS LV structure	00 _h	00 _h
84 _h	1	FWI, SFGI	72 _h	FF _h
85 _h	1	CID Support	00 _h	00 _h
86 _h	3	DATA_RATE_MAX	000000 _h	FFFF00 _h

For Type A SFGI value, coexistence issues with legacy NFC services may appear. In such cases, increasing SFGI value could be allowed, with impact only on transaction performance. In any case, this value shall not exceed 8.

Note: LSB byte of the two-byte ATQA value is transmitted first.

Table 15-7: Values for Protocol Data Type B (tag 87_h)

Tag	Length (bytes)	Value Description	Value in Protocol Parameter Data (Tag 'A0')	Value in Protocol Parameter Mandatory Mask (Tag 'A1')
80 _h	1	PUPI LV structure LV structure	00 _h	00 _h
81 _h	1	AFI	00 _h	00 _h
82 _h	4	ATQB	00000071 _h	000000F1 _h
83 _h	1	Higher Layer Response in response to ATTRIB	00 _h	00 _h
84 _h	3	Maximum data rate	000000 _h	FFFF00 _h

NOTE: For some areas, certain contactless services require fixed parameter values. In such cases, corresponding mask values as defined in [Table 15-6](#) and [Table 15-7](#) may be adapted to allow availability of these services concurrently with the DK service.

15.3.1.9 Notification of the Digital Key framework

Notification for APDU commands processed over the contactless interface

This notification mechanism may be used in case the APDU command is processed over the contactless interface. The DK applet uses the Connectivity Service interface defined in ETSI TS 102 705 [25] to request the NFC Controller to create a Host Controller Interface (HCI) event EVT_TRANSACTION (as defined in ETSI TS 102 622 [26]) as follows:

- The field AID (Tag 81_h) contains the AID of the DK applet.
- The field PARAMETERS (Tag 82_h) contains a list of notifications. The coding of each notification is as defined in [Table 15-9](#). These notifications are appended to the list in the sequential order they are generated.

Using this notification mechanism is referred to as Option C in this specification.

Notification for APDU commands processed over the wired interface

This notification mechanism may be used in case the APDU command is processed over the wired interface. The DK framework may issue a GET NOTIFICATION command described in Section [15.3.2.28](#) to retrieve the notification(s) potentially generated by the DK applet during the processing of the previous APDU command.

Using this notification mechanism is known as Option D in this specification.

Table 15-8: Status Word— Command successfully executed

SW as defined in the listing in Section 15.3.2	SW sent by the Digital Key applet to the Digital Key framework	Behavior of Digital Key framework
SW 9000 _h	SW 9000 _h if no notification is pending	Forward SW 9000 _h to the vehicle
	SW 9100 _h if a notification is pending	Forward SW 9000 _h to the vehicle

SW as defined in the listing in Section 15.3.2	SW sent by the Digital Key applet to the Digital Key framework	Behavior of Digital Key framework
		Issue a GET NOTIFICATION command to retrieve the notification(s)
All other SWs	SW as defined in the listing (notification pending or not)	Forward the received SW to the vehicle Issue a GET NOTIFICATION command to retrieve potentially generated notification(s)

Coding of the notifications

Table 15-9: Coding of the notification

Tag	Length (bytes)	Description	Field is
7F60 _h	variable	Notification	mandatory
81 _h	1	Notification code as defined in Table 15-11	mandatory
A0 _h	variable	Notification context as defined in Table 15-10	mandatory
8E _h	variable	Additional data as defined in Table 15-11	conditional

The notification context is defined in [Table 15-10](#).

The different values of the notification code corresponding to the different notification notify_* are defined in [Table 15-11](#).

The presence and content of the additional data depends on the notification code (see [Table 15-11](#)).

Table 15-10: Notification Context

Tag	Length (bytes)	Description	Field is
A0 _h	variable	Notification context	mandatory
50 _h	20	key_identifier, SHA-1 hash of the value of the BIT STRING subjectPublicKey of the target endpoint (excluding the tag, length, and number of unused bits)	conditional
4D _h	8	vehicle_identifier	conditional

The presence of the key_identifier and of the vehicle_identifier depends on the notification code (see [Table 15-11](#)).

Table 15-11: Value and Presence of the Different Fields in the HCI Notification Event

Notification	Code	Field key_identifier is	Field vehicle_identifier is	Field additional data is
notify_start_of_transaction	00 _h	NA	NA	NA
notify_endpoint_not_found_fast	01 _h	NA	mandatory	Length: 2

Notification	Code	Field key_identifier is	Field vehicle_identifier is	Field additional data is
				Value: P1, P2 of AUTH0 command
notify_endpoint_not_found_standard	02 _h	NA	mandatory	Length: 2 Value: P1, P2 of AUTH0 command
notify_user_authentication_not_performed	03 _h	NA	mandatory	Length: 2 Value: P1, P2 of AUTH0 command
notify_standard with nvm.endpoint.identifier	04 _h	mandatory	mandatory	Length: 2 Value: P1, P2 of AUTH0 command
notify_fast_kpersistent_established with nvm.endpoint.identifier	05 _h	mandatory	mandatory	Length: 2 Value: P1, P2 of AUTH0 command
notify_fast_kpersistent_not_established with nvm.endpoint.identifier	06 _h	mandatory	mandatory	Length: 2 Value: P1, P2 of AUTH0 command
notify_vehicle_authentication_failed	07 _h	optional	optional	NA
notify_vehicle_authentication_success	08 _h	optional	optional	NA
notify_endpoint_not_found_presence	09 _h	NA	mandatory	NA
notify_presence with nvm.endpoint.identifier	0A _h	mandatory	mandatory	NA
notify_mailbox_written	0C _h	optional	optional	NA
notify_deselect with reason	0D _h	NA	NA	Length: variable Value: Reason
notify_end_of_transaction with P1, P2 during a fast or standard transaction	0E _h	optional	optional	Length: 4 Value: P1, P2 of CONTROL FLOW command, P1, P2 of AUTH0 command
notify_end_of_transaction with P1, P2 outside a fast or standard transaction	0E _h	optional	optional	Length: 4

Notification	Code	Field key_identifier is	Field vehicle_identifier is	Field additional data is
				Value: P1, P2 of CONTROL FLOW command, FFFF _h
notify_application_specific with P1, P2 during a fast or standard transaction	0F _h	optional	optional	Length: 4 Value: P1, P2 of CONTROL FLOW command, P1, P2 of AUTH0 command
notify_application_specific with P1, P2 outside a fast or standard transaction	0F _h	optional	optional	Length: 4 Value: P1, P2 of CONTROL FLOW command, FFFF _h
notify_message_in_exchange	10 _h	optional	optional	Length: variable Value: Data present in the value part of tag 8E _h of the EXCHANGE command
notify_URSK_created	11 _h	optional	NA	NA
RFU	12 _h –FF _h	-	-	-

15.3.2 Commands

15.3.2.1 SELECT command

The AID of the Digital Key applet shall be A0000008094343444B417631_h.

This command selects the Digital Key applet instance to be used for the transaction. The instance AID parameter is defined during applet installation (see the INSTALL for INSTALL command in [20]).

The SE should respond with the status word 6985_h if the Digital Key applet is temporarily unavailable, in which case the vehicle should retry application.

The device may also respond with the status word 6A82_h if the route to the Digital Key applet is not yet properly configured in the NFC Controller between Owner Pairing Phase 2 and Phase 3. If the vehicle receives the 6A82_h status word during the Owner Pairing procedure it should retry applet selection as described in section 6.3.4.

1 **command:** CLA1 A4 04 00 Lc [instance AID] 00

2 **response:** [Table 15-12] 90 00

3 The CLA1 is as defined in Table 15-3.

4 Table 15-12: SELECT Response Fields

Tag	Length (bytes)	Description	Field is	Domain Version
5C _h	variable	A list of supported Digital Key applet transaction (V-D-TX) protocol versions (ver.high ver.low) ordered from highest to lowest. Each version number is concatenated and encoded on 2 bytes in big-endian notation.	mandatory	V-D-TX

5 Listing 15-2: SELECT Processing

```
1 input : none
2 output : supported_protocol_versions_tlv
3 begin
4   cod.transaction_state ← select_done
5   cod.atomic_session ← stopped
6   if contactless interface
7     sent to digital key framework notify_start_of_transaction
7   return nvm.supported_protocol_versions_tlv
8 end
```

6 15.3.2.2 CREATE CA command

7 This command creates an onboard signing authority called Instance CA used for endpoint
8 issuance. The generation and provisioning of this entity is out of scope of this specification. The
9 certificate containing the Instance CA public key shall comply with Listing 15-16. The INS
10 value for CREATE CA command shall be 38_h.

11 15.3.2.3 DELETE CA command

12 This command deletes an Instance CA. The method of deletion of this entity is out of scope of
13 this specification. The INS value for DELETE CA command shall be 3A_h.

14 15.3.2.4 CREATE ENDPOINT command

15 The command creates a communication endpoint and provides the corresponding endpoint
16 certificate. A communication endpoint distributes signatures and mailbox content over a secure
17 channel to authenticated parties only.

18 The endpoint configuration fields described in Table 15-13 may be provided either in the
19 command payload or in the internal buffer using the WRITE BUFFER command described in
20 Section 15.3.2.14. The endpoint creation certificate described in Listing 15-5 is always provided
21 in the internal buffer and is accessible using the READ BUFFER command described in Section
22 15.3.2.13.

23 **command:** CLA2 70 00 00 Lc ([Table 15-13] [Table 15-17] 00)

24 **response:** [response_length] 9000

25 The CLA2 is as defined in Table 15-3.

1

Table 15-13: Endpoint Configuration

Tag		Length (bytes)	Description	Field is	Domain Version
7F27 _h		variable	Endpoint configuration for device supporting sharing in a chain in this Digital Key Specification.	mandatory	V-OD-FW (Owner Pairing) OD-FD-KS (Key Sharing)
	4D _h	8	vehicle_identifier	mandatory	V-OD-FW
	5F20 _h	1–30	endpoint_identifier, DER encoded PrintableString as per RFC 5280 [3] (without PrintableString tag and length)	mandatory	V-OD-FW
	42 _h	1–30	instance_CA_identifier or intermediate_CA_identifier , DER encoded PrintableString as per RFC 5280 [3] (without PrintableString tag and length)	mandatory	V-OD-FW
	46 _h	1	option_group_1 (on bits 0–7): bit0 : enable(1)/disable(0) Standard transaction allowed on contactless interface bit1 : enable(1)/disable(0) Fast transaction allowed on contactless interface bit2 : enable(1)/disable(0) Standard transaction allowed on wired interface bit3 : enable(1)/disable(0) Fast transaction allowed on wired interface bit4 : enable(1)/disable(0) AUTHORIZE ENDPOINT command allowed bit5 : enable(1)/disable(0) Authorizing endpoints which have the AUTHORIZE ENDPOINT command allowed bit6 : enable(1)/disable(0) EXCHANGE command allowed on wired interface bit7 : enable(1)/disable(0) EXCHANGE command allowed after AUTH0	mandatory	V-OD-FW
	47 _h	1	option_group_2 (on bits 0–7): bit0 : enable(1)/disable(0) SIGN command allowed bit1 : enable(1)/disable(0) Exported data from confidential mailbox are replaced with random values bit2 : key class: key issued by mobile device (0) or server (1) bit3 : key class: key issued for a private vehicle (0) or fleet vehicle (1) bit4 : RFU (shall be set to 0) bit5 : RFU (shall be set to 0) bit6 : enable (1)/disable (0), compressed endpoint PK format in attestation package	mandatory	V-OD-FW

Tag		Length (bytes)	Description	Field is	Domain Version
			bit7: enable(1)/disable(0) SET CONFIDENTIAL DATA command allowed		
	5C _h	2	protocol_version	mandatory	V-OD-FW
	5B _h	65	vehicle_PK prepended by 04 _h	mandatory	V-OD-FW
	51 _h	15 or 13	not_before, DER encoded GeneralizedTime (15 bytes in length) or UTCTime (13 bytes in length) as per RFC 5280 [2]	mandatory	V-OD-FW OD-FD-KS
	52 _h	15 or 13	not_after, DER encoded GeneralizedTime (15 bytes in length) or UTCTime (13 bytes in length) as per RFC 5280 [3]	mandatory	V-OD-FW OD-FD-KS
	53 _h	2	Endpoint_conversion_counter	optional	V-OD-FW
	49 _h	variable	authorized_PK[], list of up to 5 public keys prepended by 04 _h of 65 bytes length	optional	V-OD-FW
	4A _h	2	confidential_mailbox_size	optional	V-OD-FW
	4B _h	2	private_mailbox_size	optional	V-OD-FW
	4E _h	1–8	key_slot	optional	D-VS
	5F56 _h	variable	account_info_hash	optional	V-OD-FW

vehicle_identifier: Field provided by the endpoint creator; this field is used by the applet for fast endpoint lookup during transaction.

endpoint_identifier: Arbitrary field provided by the endpoint creator, describing the subject of the certificate. The endpoint_identifier field typically identifies the entity associated with the public key present in the certificate.

instance_CA_identifier/intermediate_CA_identifier: This field allows for selection of the instance CA used by the SE to sign the endpoint creation certificate. For servers it allows for selection of an intermediate CA used to sign the endpoint creation certificate. This value is present in the field “subject CommonName” of the corresponding instance/intermediate certificate.

endpoint_conversion_counter: This field shall be omitted if the endpoint is created in V-OD-FW < 0300_h. When this field is omitted, endpoint_conversion_counter is set to 0000_h by default. This field shall be present and set to 0001_h if the endpoint is created in V-OD-FW = 0300_h. In

- 1 Convert Endpoint command, target_endpoint_conversion_counter is compared to
- 2 endpoint_conversion_counter to determine whether the endpoint can be converted or not.
- 3 **option_group_1 and option_group_2:** For Owner Pairing (Owner Pairing Phase 2), the vehicle
- 4 shall set these two fields according to [Table 15-14](#).

5 *Table 15-14: Setting of Tag 46_h and Tag 47_h by vehicle for various wireless capability combinations.*

Tag	Value	Description
Vehicle supporting WCC1		
46 _h	bit0 set to 1	Standard transaction allowed on contactless interface
	bit1 set to 1	Fast transaction allowed on contactless interface
	bit2 set to 0	Standard transaction not allowed on wired interface
	bit3 set to 0	Fast transaction not allowed on wired interface
	bit4 set to 1	AUTHORIZE ENDPOINT command allowed
	bit5 set to 1	Authorizing endpoints which have the AUTHORIZE ENDPOINT command allowed
	bit6 set to 0	EXCHANGE command not allowed on wired interface
	bit7 set to 0/1	EXCHANGE command allowed (1) or not allowed (0) after AUTH0
Vehicle supporting WCC2 or WCC3		
46 _h	bit0 set to 1	Standard transaction allowed on contactless interface
	bit1 set to 1	Fast transaction allowed on contactless interface
	bit2 set to 1	Standard transaction allowed on wired interface
	bit3 set to 0	Fast transaction not allowed on wired interface
	bit4 set to 1	AUTHORIZE ENDPOINT command allowed
	bit5 set to 1	Authorizing endpoints which have the AUTHORIZE ENDPOINT command allowed
	bit6 set to 1	EXCHANGE command allowed on wired interface
	bit7 set to 0/1	EXCHANGE command allowed (1) or not allowed (0) after AUTH0
47 _h	bit0 set to 1	SIGN command allowed
	bit1 set to 1	Exported data from confidential mailbox are replaced with random values
	bit2 set to 0/1	Sender class (see Table 15-16)
	bit3 set to 0/1	Vehicle class (see Table 15-16)
	bit4 set to 0	RFU
	bit5 set to 0	RFU
	bit6 set to 0/1	Endpoint PK format in attestation package supported by the vehicle: (0) uncompressed only, (1) compressed or uncompressed.
	bit7 set to 0/1	SET CONFIDENTIAL DATA command not allowed (0) if immobilizer tokens are not used or allowed (1) if immobilizer tokens are used

6

The bit7 of tag 46_h, shall not be set to 1 if sensitive data such as immobilizer tokens are exchanged using the EXCHANGE command.

The vehicle can request in bit6 of tag 47_h the endpoint PK in the attestation package (see [Table 11-13](#)) to be provided in compressed format. Devices can provide the compressed format, the vehicles shall support both, compressed and uncompressed options even if it indicates support for compressed format. The goal of supporting compressed format is to decrease the size of the attestation package and transmission time in the First New Key Transaction.

For key sharing (Key Sharing flow step 1 and 2), the owner device shall set these two fields which is within Key Creation Request according to [Table 15-15](#).

Table 15-15: Setting of Tag 46_h and Tag 47_h by the owner device for the key sharing.

Tag	Value	Description
46 _h	bit0 to bit7	As defined in the endpoint configuration (owner device)
47 _h	bit0 to bit7	As defined in the endpoint configuration (owner device)

The Key Class (bit 2 and bit3) defines whether a key is issued by a mobile device or a server and whether the key is issued for a private vehicle, fleet vehicle or unpaired vehicle. A private vehicle is a vehicle that contains an owner key or at least one shared key with owner role. A fleet vehicle is a vehicle that is paired (in-fleeted) by a key server (SBOD) and not unpaired (de-fleeted). An unpaired vehicle is a vehicle that is neither private vehicle nor fleet vehicle, e.g., vehicle during production phase, vehicle during transport to dealership, etc.

An SBOD can issue keys for a fleet vehicle. A SBFD can issue delegate keys for a private vehicle and for a fleet vehicle. A KIS server can issue keys for a private vehicle, fleet vehicle, and unpaired vehicle.

Table 15-16: Tag 47_h indicating Key Class

Tag	Value	Description
47 _h	bit2	0: shared key from mobile device 1: shared key from SBxD/KIS
	bit3	0: shared key for private vehicle (legacy) 1: shared key for fleet or unpaired vehicle

protocol_version: Digital Key applet protocol transaction (V-D-TX) version assigned at the time of the endpoint creation. This field is informative and shall not be used by the applet to enforce usage of a specific protocol version. Thus, this shall not prevent the endpoint to be used with other versions of the Digital Key applet protocol version that may be supported by the vehicle and the Digital Key applet later.

vehicle_PK: The public key of the vehicle.

Not_before: The issued certificate start of validity date. The validity date is not required to be verified by the applet.

Not_after: The issued certificate end of validity date. The validity date is not required to be verified by the applet.

Authorized_PK[]: The public keys used by the device to verify other SE's endpoint creation certificates. This field shall be present, if number of shareable levels for the endpoint is greater than 0 (See Section 2.8.4)

Confidential_mailbox_size: If field not present, or if present with confidential_mailbox_size equal to 0000_h, confidential mailbox is not available.

Private_mailbox_size: If field not present, or if present with private_mailbox_size equal to 0000_h, private mailbox is not available.

Key_slot: A field in the endpoint configuration that is populated in the receiver endpoint configuration during step 1 of the key sharing (see Table 11-5) if slot identifiers are retrieved from the vehicle via the owner device. The value of this field is set to one of the slot identifiers from the SlotIdentLst field. The key_slot can be manually set when a fast search among a large number of public keys is required on the vehicle side. The fast search can be facilitated by attributing key slot numbers instead of relying on the automatic hash-based naming during endpoint creation. If key_slot field is not provided, an automatic slot number is generated as described in Listing 15-6, since the key_slot is mandatory in the AUTH1-Response. In case of slot identifiers provided online, the owner device during key sharing, assigns an automatically generated random slot identifier. The slot identifier is used to derive the Kble_oob, see Sections 19.5.1 and 19.5.8.

Note: A device generated key_slot does not safeguard against attestation replay attacks during key sharing.

This field does not need to be unique among endpoints created on an instance.

This field can be updated after key creation with the SETUP_ENDPOINT command. The value "initial_key_slot" in the endpoint certificate will always remain the one initially assigned during endpoint creation.

account_info_hash: As defined in Section 6.3.4.3, if present this shall be added to the endpoint certificate in the endpoint certificate extension schema (Listing 15-3)

If the applet_implementation_model field is included (i.e., SE-centric applet model) in the INSTALL for INSTALL command, the following additional tags shall appear after the tag 7F27_h.

Table 15-17: Endpoint verification information

Tag	Length (bytes)	Description	Field is
7F28 _h	variable	Vehicle OEM CA Certificate (signed by Device OEM)	conditional
7F4C _h	variable	DER-encoded X.509 Intermediate Certificate	optional
7F4B _h	variable	DER-encoded X.509 Vehicle Public Key Certificate	conditional

Vehicle OEM CA Certificate (signed by Device OEM): this field is provided by the framework and is verified by the SE. The SE also uses this data to verify the Vehicle Public Key Certificate.

It is assumed that Device OEM CA root certificate is stored in the SE for this operation.

Vehicle Public Key Certificate: This field is provided by the vehicle during owner pairing. This field is not present during Digital Key sharing. The SE shall verify the Vehicle Public Key Certificate if this field is present.

1 *Listing 15-3: Endpoint Certificate Extension Schema*

```

1 Schema DEFINITIONS EXPLICIT TAGS
2 BEGIN
3 EndpointCertificateExtensionSchema ::= SEQUENCE
4 {
5     extension_version      INTEGER (1..255),
6     vehicle_identifier      OCTET STRING (SIZE (8)),
7     option_group_1         OCTET STRING (SIZE (1)),
8     option_group_2         OCTET STRING (SIZE (1)),
9     protocol_version        OCTET STRING (SIZE (2)),
10    vehicle_PK              PublicKey,
11    initial_key_slot         OCTET STRING (SIZE (1..8)),
12    authorized_PK_list       SEQUENCE(SIZE(1..5)) OF PublicKey OPTIONAL,
13    confidential_mailbox_size [0] INTEGER (1..65535) OPTIONAL,
14    private_mailbox_size     [1] INTEGER (1..65535) OPTIONAL,
15    account_info_hash        OCTET STRING (SIZE (32)) OPTIONAL
16 }
17
18 PublicKey ::= SEQUENCE
19 {
20     algorithm      AlgorithmIdentifier,
21     public_key     BIT STRING
22 }
23
24 AlgorithmIdentifier ::= SEQUENCE
25 {
26     algorithm      OBJECT IDENTIFIER,
27     parameters    ANY DEFINED BY algorithm OPTIONAL
28 }
29 END

```

2 *Listing 15-4: Endpoint Certificate Extension Data*

```

1 endpoint-cert-extension-data EndpointCertificateExtensionSchema ::=
2 {
3     extension_version      1, --value indicating the extension version, this value is meant to be incremented if the extension fields
4     next applet versions.
5     vehicle_identifier      '...'H, --value as per endpoint configuration
6     option_group_1         '...'H, --value as per endpoint configuration
7     option_group_2         '...'H, --value as per endpoint configuration
8     protocol_version        '...'H, --value as per endpoint configuration
9     vehicle_PK
10    {
11        algorithm
12        {
13            algorithm {1 2 840 10045 3 1 7} --OID for prime256v1(ANSI X9.62 named elliptic curve)
14        },
15        public_key '04...'H --vehicle_PK pre-pended with 04n
16    },
17    initial_key_slot      '...'H, --value as per endpoint configuration
18    authorized_PK_list
19    {
20        {
21            algorithm
22            {
23                algorithm {1 2 840 10045 3 1 7} --OID for prime256v1(ANSI X9.62 named elliptic curve)
24            },

```

```

25 public_key '04...'H --authorized_PK
26 },
27 ... --other authorized_PK
28 },
29 confidential_mailbox_size ..., --value as per endpoint configuration
30 private_mailbox_size      ..., --value as per endpoint configuration
31 }

```

1

Listing 15-5: Endpoint Certificate Data

```

1 endpoint-cert-data Certificate ::=
2 {
3   tbsCertificate
4   {
5     version v3, --shall be v3
6     serialNumber ..., --a random integer chosen by the certificate issuer
7     signature
8     {
9       algorithm {1 2 840 10045 4 3 2} --OID for ecdsaWithSHA256 (ANSI X9.62 ECDSA algorithm with SHA256)--
10    },
11    issuer rdnSequence:
12    {
13      {
14        {
15          type {2 5 4 3}, --OID for CommonName
16          value "..." --the instance_CA_identifier as per endpoint configuration
17            -- (PrintableString or UTF8String recommended)
18        }
19      }
20    },
21    validity
22    {
23      notBefore Time: "...", --value not_before as per endpoint configuration
24      notAfter Time: "...", --value not_after as per endpoint configuration
25    },
26    subject rdnSequence:
27    {
28      {
29        {
30          type {2 5 4 3}, --OID for CommonName
31          value "..." --the endpoint_identifier as per endpoint configuration, shall use PrintableString or UTF8String format
32        }
33      }
34    },
35    subjectPublicKeyInfo
36    {
37      algorithm
38      {
39        algorithm {1 2 840 10045 2 1}, --OID for ecPublicKey (ANSI X9.62 public key type)
40        parameters {1 2 840 10045 3 1 7} --OID for prime256v1 (ANSI X9.62 named elliptic curve)
41      },
42      subjectPublicKey '04...'H --the public key pre-pended with 04 to indicate uncompressed format
43    },
44    extensions
45    {
46      {
47        extnID {install_param_oid_endpoint}, --OID for Endpoint certificate
48        critical TRUE,

```

```

50     extnValue  '...'H --DER encoding for endpoint-cert-extension-data as per Listing 15-4
51   },
52   {
53     extnID     {2 5 29 15}, --OID for KeyUsage standard extension
54     critical   TRUE,
55     extnValue  '03020780'H --DER encoding for KeyUsage, digitalSignature
56   },
57   {
58     extnID     {2 5 29 19}, --OID for BasicConstraints standard extension
59     critical   TRUE,
60     extnValue  '3000'H --DER encoding for CA=FALSE
61   },
62   {
63     extnID     {2 5 29 35}, --OID for AuthorityKeyIdentifier standard extension
64     critical   FALSE,
65     extnValue  '...'H -- DER encoding of an AuthorityKeyIdentifier sequence as defined in \[3\], containing only
66                 – a KeyIdentifier element. The KeyIdentifier is an OCTET STRING containing the
67                 – 160-bit SHA-1 hash of the value of the BIT STRING subjectPublicKey
68                 – from the issuer certificate (excluding the tag, length, and number of unused bits)
69   },
70   ...
71 }
72 },
73 signatureAlgorithm
74 {
75   algorithm {1 2 840 10045 4 3 2}
76 },
77 signatureValue '...'H --the certificate signature computed as per RFC 5280 \[3\].
78 }

```

1

Listing 15-6: CREATE ENDPOINT Processing

```

1  input: Table 15-13, Table 15-17
2  output: response_length, certificate
3  begin
4    if no payload present in command
5      if Table 15-13 data fields not present in cod.internal_buffer
6        return 6400h
7    if any mandatory field is not present
8      return 6400h
9    if endpoint_identifier already exists in any of the nvm.endpoint[].identifier
10     return 6400h
11   if protocol_version from input table is not supported
12     return 6400h
13   if instance CA referenced by instance_CA_identifier field does not exist
14     return 6400h
15   if vehicle_identifier is already used by any of the existing of the nvm.endpoint[]
16     return 6400h
17   if SE-centric applet model
18     if Table 15-17 data fields not present in cod.internal_buffer
19       return 6400h
20     if Table 15-17 data fields are not verified
21       return 6400h
22   generate key pair endpoint_PK, endpoint_SK according to Listing 15-41
23   if key_slot field not provided
24     generate key_slot as per Listing 15-42 using subjectPublicKey of the endpoint to be created as input (excluding the tag,
length, and number of unused bits)
25   generate certificate using Listing 15-5 as per RFC 5280 \[3\]
26   atomic start

```

```

27  nvmwrite nvm.endpoint.<...> configuration as per Table 15-13 and certificate parts (including certificate signature) needed to
    re-generate the certificate when retrieved with VIEW command
28  nvmwrite nvm.endpoint.identifier ← endpoint_identifier
29  nvmwrite nvm.endpoint.key_slot ← key_slot
    nvmwrite nvm.endpoint.initial_key_slot ← key_slot
30  nvmwrite nvm.endpoint.PK ← endpoint_PK
31  nvmwrite nvm.endpoint.SK ← endpoint_SK
32  nvmwrite nvm.endpoint.Kpersistent, fill with randoms
33  nvmwrite nvm.endpoint.kpersistent_established ← false
34  nvmwrite nvm.endpoint.terminated ← false
35  nvmwrite nvm.endpoint.transaction_code_list_cless ← []
36  nvmwrite nvm.endpoint.transaction_code_list_wired ← []
37  atomic commit

38  store certificate in cod.internal_buffer
39  response_length ← length written in cod.internal_buffer
40  return response_length (2 bytes big-endian)
41  end

```

1 15.3.2.5 TERMINATE ENDPOINT command

2 This command sets the endpoint in terminated state and returns a termination attestation. If the
3 endpoint is already in terminated state, the previously computed attestation is returned. The
4 terminated state of an endpoint cannot be reverted; the only possible action is to delete the
5 endpoint.

6 The following command formats are allowed:

- 7 • If no secure channel is to be established between the Digital Key framework and the Digital
8 Key applet, then either command format 1 or command format 2 may be used and
9 implemented by the applet.
- 10 • If a secure channel is to be established between the Digital Key framework and the Digital
11 Key applet (option A), command format 2 shall be used and shall be implemented by the
12 applet.

13 Command format 1:

14 **command:** CLA2 72 00 00 Lc [[Table 15-18](#)] 00

15 **response:** [[Table 15-20](#)] 9000

16 The CLA2 is as defined in [Table 15-3](#).

17 Command format 2:

18 **command:** CLA2 72 00 00 Lc [[Table 15-18](#)] 00

19 **response:** [response_length] 9000

20 The CLA2 is as defined in [Table 15-3](#).

21 *Table 15-18: Endpoint Termination Request*

Tag	Length (bytes)	Description	Field is	Domain Version
7F2E _h	variable	Endpoint Termination Request	mandatory	N/A
50 _h	20	key_identifier, SHA-1 hash of the value of the BIT STRING subjectPublicKey of the target endpoint (excluding the tag, length, and number of unused bits)	mandatory	V-OD-FW

Tag	Length (bytes)	Description	Field is	Domain Version
91 _h	16	termination_requester_nonce, a random value chosen by the termination requester	mandatory	N/A
58 _h	1–40	arbitrary_data, arbitrary data field chosen by the termination requester	optional	N/A

1 *Table 15-19: Endpoint Termination Attestation Data Fields*

Tag	Length (bytes)	Description	Field is	Domain Version
41 _h	1	version = 01 _h , version of Endpoint Termination Attestation	mandatory	Informative only. N/A
92 _h	8	random	mandatory	D-VS
5F20 _h	1–30	endpoint_identifier, identifier of the terminated endpoint	mandatory	V-OD-FW
5F49 _h	65	the public key of the terminated endpoint	mandatory	V-OD-FW
91 _h	16	termination_requester_nonce	mandatory	D-VS
58 _h	1–40	arbitrary_data	optional	D-VS
93 _h	4	usage = D6854CB9 _h	mandatory	D-VS

2 *Table 15-20: Endpoint Termination Attestation*

Tag	Length (bytes)	Description	Field is	Domain Version
7F29 _h	variable	Endpoint Termination Attestation	mandatory	D-VS
Content of Table 15-19			mandatory	
9E _h	64	signature with the private key of the terminated endpoint over fields from Table 15-19	mandatory	D-VS

3 *Listing 15-7: TERMINATE ENDPOINT Processing for Command Format 1*

```

1  input: key_identifier
2  output: termination_attestation
3  begin
4    if key_identifier does not match with any of the existing endpoints
5      return 6400h
6    if nvm.endpoint.terminated = true
7      return nvm.endpoint.termination_attestation
8    if UWB present
9      delete URSK associated to the key_identifier
10   nvm.endpoint.terminated ← true
11   generate 8 bytes random as per Listing 15-40
12   generate signature over Table 15-19 using key selected from nvm.endpoint.SK according to Listing 15-43
13   nvmwrite nvm.endpoint.termination_attestation ← data structure as per Table 15-20
14   return nvm.endpoint.termination_attestation
15 end

```

4 *Listing 15-8: TERMINATE ENDPOINT Processing for Command Format 2*

```

1  input: key_identifier
2  output: response length, termination_attestation
3  Begin
4    if key_identifier does not match with any of the existing endpoints
5      return 6400h

```

```

6  if nvm.endpoint.terminated = true
7      store nvm.endpoint.termination_attestation in cod.internal_buffer
8      response_length ← length written in cod.internal_buffer
9      return response_length (2 bytes big-endian)
10 if UWB present
11     delete URSK associated to the key_identifier
12     nvm.endpoint.terminated ← true
13     generate 8 bytes random as per Listing 15-40
14     generate signature over Table 15-19 using key selected from nvm.endpoint.SK according to Listing 15-43
15     nvmmwrite nvm.endpoint.termination_attestation ← data structure as per Table 15-20
16     store nvm.endpoint.termination_attestation in cod.internal_buffer
16     response_length ← length written in cod.internal_buffer
18     return response_length (2 bytes big-endian)
19 End

```

1 15.3.2.6 DELETE ENDPOINT command

2 This command will delete an endpoint and release the associated memory.

3 **command:** CLA2 74 00 00 Lc [Table 15-21]

4 **response:** 9000

5 The CLA2 is as defined in Table 15-3.

6 Table 15-21: Deletion Request

Tag	Length (bytes)	Description	Field is	Domain Version
50 _h	20	key_identifier, SHA-1 hash of the value of the BIT STRING subjectPublicKey of the target endpoint (excluding the tag, length, and number of unused bits)	mandatory	V-OD-FW

7 Listing 15-9: DELETE ENDPOINT Processing

```

1  input: key_identifier
2  output: none
3  begin
4      if key_identifier does not match with any of the existing endpoints
5          return 6400h
6      if UWB present
7          delete URSK associated to the key_identifier
8      atomic start
9          nvmdeldelete nvm.endpoint
10     atomic commit
11 end

```

8 15.3.2.7 AUTHORIZE ENDPOINT command

9 This command signs an endpoint public key (issued by a receiver device) and an arbitrary
10 additional data field using a local endpoint private key (sender device). Optionally, the command
11 can also extract and encrypt a portion of the confidential mailbox to be inserted in the authorized
12 recipient endpoint.

13 Input and output buffers are transferred using the internal buffer commands READ BUFFER and
14 WRITE BUFFER to avoid APDU length limitations.

The supported verification chains are described in Section 16.6. Chain verification starts from one of the authorized public keys stored in the endpoint and continues from top to bottom with the items in Table 15-23.

This command requires user authentication to be performed on device.

This command can generate the attestation data in different versions.

Note: The protocol_version value present in the receiver’s endpoint certificate shall not be checked during processing of this command.

Depending on the selected P2 parameter of the command, the attestation data format is different. This can be used by the caller to generate attestation versions supported by the vehicle. In this Digital Key specification, the following values for P2 are allowed. All other values shall be rejected with SW = 6A86h.

- P2 = 00h, generate attestation data as per Table 15-24 (values for 01h) – legacy support
- P2 = 01h, generate attestation data as per Table 15-24 (values for 01h)
- P2 = 03h, generate attestation data as per Table 15-25 (values for 03h)

The endpoint public key can be in uncompressed or compressed format. Uncompressed format is prepended with 04h, compressed format is prepended with 02h (if y is even) and 03h (if y is odd).

command: CLA2 32 00 P2 Lc [Table 15-22] 00

response: [response_length] 90 00

Table 15-22: AUTHORIZE ENDPOINT Command Payload

Tag	Length (bytes)	Description	Field is	Domain Version
50h	20	key_identifier, SHA-1 hash of the value of the BIT STRING subjectPublicKey of the target endpoint (excluding the tag, length, and number of unused bits)	mandatory	V-OD-FW
4Ah	4	offsetmsb offsetlsb lengthmsb lengthlsb, portion of the confidential mailbox to be exported	optional	V-OD-FW
58h	variable	arbitrary_data	optional	N/A

Listing 15-10: External CA Certificate Extension Schema

```
ExternalCACertificateExtensionSchema ::= SEQUENCE
{
    extension_version    INTEGER (1..255)
}
```

Listing 15-11: External CA Certificate Extension Data

```
external-ca-cert-extension-data ExternalCACertificateExtensionSchema ::=
{
    extension_version 1    --value shall be 1
}
```

Listing 15-12: External CA Certificate Basic Constraints Extension Data

```
external-ca-cert-basic-constraints BasicConstraints ::=
{
    CA                TRUE,
```

```
4   pathLenConstraint ...    --value shall be 1 if an Instance CA is conditionally or always present in the chain
5                           --value shall be 0 if Instance CA is never present in the chain
6   }
```

1 *Listing 15-13: External CA Certificate Data*

```
1 external-ca-cert-data Certificate ::=
2 {
3   tbsCertificate
4   {
5     version v3, --shall be v3
6     serialNumber ..., --a random integer chosen by the certificate issuer
7     signature
8     {
9       algorithm {1 2 840 10045 4 3 2} --OID for ecdsaWithSHA256 (ANSI X9.62 ECDSA algorithm with SHA256)
10    },
11    issuer rdnSequence:
12    {
13      {
14        {
15          type {2 5 4 3}, --OID for CommonName
16          value "...", --shall match the subject of the issuer certificate
17            --(shall use PrintableString or UTF8String format)*
18        }
19      }
20    },
21    validity
22    {
23      notBefore Time: "...", -- shall use UTCTime or GeneralizedTime as defined in \[3\]
24      notAfter Time: "...", -- shall use UTCTime or GeneralizedTime as defined in \[3\]
25    },
26    subject rdnSequence:
27    {
28      {
29        {
30          type {2 5 4 3}, --OID for CommonName
31          value "...", --contains the subject of the certificate
32            --(shall use PrintableString or UTF8String format)*
33        }
34      }
35    },
36    subjectPublicKeyInfo
37    {
38      algorithm
39      {
40        algorithm {1 2 840 10045 2 1}, --OID for ecPublicKey (ANSI X9.62 public key type)
41        parameters {1 2 840 10045 3 1 7} --OID for prime256v1(ANSI X9.62 named elliptic curve)
42      },
43      subjectPublicKey '04...'H --the public key pre-pended with 04 to indicate uncompressed format
44    },
45    extensions
46    {
47      {
48        extnID {1.3.6.1.4.1.41577.5.2}, --OID for external CA Certificate (see Appendix B.2.2)
49        critical TRUE,
50        extnValue '...'H --DER encoding for ExternalCACertificateExtensionSchema extension as per Listing 15-11
51      },
52      {
53        extnID {2 5 29 15}, --KeyUsage standard extension
```



```

54     critical   TRUE,
55     extnValue '03020204'H --DER encoding for KeyUsage, CertSign only
56   },
57   {
58     extnID     {2 5 29 19}, --BasicConstraints standard extension
59     critical   TRUE,
60     extnValue '...'H --DER encoding for BasicConstraints extension as per Listing 15-12
61   },
62   {
63     extnID     {2 5 29 35}, --OID for AuthorityKeyIdentifier standard extension
64     critical   FALSE,
65     extnValue '...'H -- DER encoding of an AuthorityKeyIdentifier sequence as defined in [3], containing only
66                   -- a KeyIdentifier element. The KeyIdentifier is an OCTET STRING containing the
67                   -- 160-bit SHA-1 hash of the value of the BIT STRING subjectPublicKey
68                   -- from the issuer certificate (excluding the tag, length, and number of unused bits)
69   },
70   {
71     extnID     {2 5 29 14}, --OID for SubjectKeyIdentifier standard extension
72     critical   FALSE,
73     extnValue '...'H --160-bit SHA-1 hash of the value of the BIT STRING subjectPublicKey
74                   --(excluding the tag, length, and number of unused bits)
75   },
76   {
77     extnID     {...}, --Optional extensions added by the issuer, content not verified by the applet
78     critical   FALSE,
79     extnValue '...'H --DER encoding of the extension
80   },
81   ...
82 }
83 },
84 signatureAlgorithm
85 {
86   algorithm {1 2 840 10045 4 3 2} --OID for ecdsaWithSHA256 (ANSI X9.62 ECDSA algorithm with SHA256)
87 },
88 signatureValue '...'H --the certificate signature by the issuer computed as per RFC 5280 [3]
89 --ECDSA signature
90 }

```

* The choice of format between PrintableString and UTF8String in the External CA certificate impacts the format of fields in Vehicle and Device OEM certificates as outlined in Section A.4.7 of Appendix A and Section B.2.5 of Appendix B

1 *Listing 15-14: Instance CA Certificate Extension Schema*

```

1 InstanceCACertificateExtensionSchema ::= SEQUENCE
2 {
3   extension_version      INTEGER (1..255),
4   applet_version         OCTET STRING (SIZE 4),
5   platform_information    OCTET STRING (SIZE (1..20)) OPTIONAL
6 }

```

2 *Listing 15-15: Instance CA Certificate Extension Data*

```

1 instance-ca-cert-extension-data InstanceCACertificateExtensionSchema ::=
2 {
3   extension_version      1, --value shall be 1
4   applet_version         '...'H --proprietary applet version information
5   platform_information    '...'H --optional proprietary platform information
6                           --field typically set by Device OEM Server to
7                           --identify the Secure Element platform
8 }

```

- 1 **Note:** applet_version and platform_information are immutable values assigned at the time of the
2 Instance CA certificate is issued. It might not reflect the current applet_version or
3 platform_information (e.g., in case of applet or Secure Element platform upgrade).

4 *Listing 15-16: Instance CA Certificate Data*

```
1 instance-ca-cert-data Certificate ::=
2 {
3   tbsCertificate
4   {
5     version v3, --shall be v3
6     serialNumber ..., --a random integer chosen by the certificate issuer
7     signature
8     {
9       algorithm {1 2 840 10045 4 3 2} --OID for ecdsaWithSHA256 (ANSI X9.62 ECDSA algorithm with SHA256)
10    },
11    issuer rdnSequence:
12    {
13      {
14        {
15          type {2 5 4 3}, --OID for CommonName
16          value "..." --shall match the subject of the issuer certificate
17            --(shall use PrintableString or UTF8String format)*
18        }
19      }
20    },
21    validity
22    {
23      notBefore Time: "...", --shall use UTCTime or GeneralizedTime as defined in [3]
24      notAfter Time: "..." -- shall use UTCTime or GeneralizedTime as defined in [3]
25    },
26    subject rdnSequence:
27    {
28      {
29        {
30          type {2 5 4 3}, --OID for CommonName
31          value "..." --contains the subject of the certificate (instance_CA_identifier)
32            --(shall use PrintableString or UTF8String format)
33        }
34      }
35    },
36    subjectPublicKeyInfo
37    {
38      algorithm
39      {
40        algorithm {1 2 840 10045 2 1}, --OID for ecPublicKey (ANSI X9.62 public key type)
41        parameters {1 2 840 10045 3 1 7} --OID for prime256v1(ANSI X9.62 named elliptic curve)
42      },
43      subjectPublicKey '04...'H
44      --the public key pre-pended with 04 to indicate uncompressed format
45    },
46    extensions
47    {
48      {
49        extnID {install_param_oid_instance_ca}, --OID for Instance CA extension
50        critical TRUE,
51        extnValue '...'H --DER encoding for instance-ca-cert-extension-data as per Listing 15-15
52      },
53    }
54  }
55 }
```

```

53 {
54   extnID    {2 5 29 15}, --KeyUsage standard extension
55   critical  TRUE,
56   extnValue '03020204'H --DER encoding for KeyUsage, CertSign only
57 },
58 {
59   extnID    {2 5 29 19}, --BasicConstraints standard extension
60   critical  TRUE,
61   extnValue '30060101FF020100'H -- DER encoding for BasicConstraints CA=TRUE, pathLenConstraint=0
62 },
63 {
64   extnID    {2 5 29 35}, --OID for AuthorityKeyIdentifier standard extension
65   critical  FALSE,
66   extnValue '...'H -- DER encoding of an AuthorityKeyIdentifier sequence as defined in [3], containing only
67               -- a KeyIdentifier element. The KeyIdentifier is an OCTET STRING containing the
68               -- 160-bit SHA-1 hash of the value of the BIT STRING subjectPublicKey
69               --from the issuer certificate (excluding the tag, length, and number of unused bits)
70 },
71 {
72   extnID    {2 5 29 14}, --OID for SubjectKeyIdentifier standard extension
73   critical  FALSE,
74   extnValue '...'H --DER encoding of the SubjectKeyIdentifier as defined in [3]. The SubjectKeyIdentifier is an
75               -- OCTET STRING containing the 160-bit SHA-1 hash of the value of the BIT STRING subjectPublicKey
76               -- (excluding the tag, length, and number of unused bits)
77 },
78 {
79   extnID    {...}, --Optional extensions added by the issuer, content not verified by the applet
80   critical  FALSE,
81   extnValue '...'H --DER encoding of the extension
82 },
83 ...
84 }
85 },
86 signatureAlgorithm
87 {
88   algorithm {1 2 840 10045 4 3 2}
89 },
90 signatureValue '...'H --the certificate signature by the issuer computed as per RFC 5280 [3]
91 }

```

* The selection of format between PrintableString and UTF8String is outlined in Section B.2.4 of Appendix B

1 *Table 15-23: AUTHORIZE ENDPOINT Internal Buffer Content Before Processing (offset 0)*

Tag	Length (bytes)	Description	Field is	Domain Version
7F20 _h	variable	External CA certificate as per Listing 15-13 from the receiver endpoint	mandatory	V-OD-FW
7F22 _h	variable	Instance CA Certificate as per Listing 15-16 from the receiver endpoint	mandatory	V-OD-FW
7F24 _h	variable	Endpoint Creation certificate as per Listing 15-5 from the receiver endpoint	mandatory	V-OD-FW
Encryption Key attestation as per Table 15-49 from the receiver endpoint			optional	D-VS

Table 15-24: AUTHORIZE ENDPOINT Attestation Data Fields with P2 = 01_h

Tag	Length (bytes)	Description	Field is	Domain Version
41 _h	1	version = 01 _h , the version of the data structure	mandatory	Informative
92 _h	8	random	mandatory	OD-FD-KS
5A _h	65	public key from the receiver endpoint creation certificate present in internal buffer	mandatory	OD-FD-KS
58 _h	variable	arbitrary_data, field present in command payload	optional	N/A
93 _h	4	usage = 91712C44 _h	mandatory	OD-FD-KS

Table 15-25: AUTHORIZE ENDPOINT Attestation Data Field with P2 = 03_h

Tag	Length (bytes)	Description	Field is	Domain Version
41 _h	1	version = 03 _h , the version of the data structure	mandatory	Informative
92 _h	8	random	mandatory	OD-FD-KS
5A _h	65 or 33	public key from the receiver endpoint creation certificate present in internal buffer, in uncompressed or compressed format depending on option_group_2 bit6	mandatory	OD-FD-KS
78 _h	variable	arbitrary_data, field present in command payload	optional	N/A
93 _h	4	usage = 91712C44 _h	mandatory	OD-FD-KS

Table 15-26: AUTHORIZE ENDPOINT Internal Buffer Content After Processing (offset 0)

Tag	Length (bytes)	Description	Field is	Domain Version
9E _h	64	sender endpoint signature over fields from Table 15-24	mandatory	OD-FD-KS
92 _h	8	random	mandatory	OD-FD-KS
97 _h	65	encsender_ePK	optional	OD-FD-KS
4A _h	variable	encrypted_mailbox mac	optional	OD-FD-KS

Listing 15-17: AUTHORIZE ENDPOINT Processing

```

input: internal_buffer, key_identifier(, mailbox_offset, mailbox_length, arbitrary_data)
output: response_length, signature(, encrypted_mailbox, mac, encsender_ePK)
begin
  if key_identifier does not match with any existing endpoint or target endpoint is terminated
    return 6A88h
  if AUTHORIZE ENDPOINT is not allowed on selected sender endpoint as per Table 15-13
    return 6900h
  if ua_performed != true -- ua_performed true if user authentication performed, false otherwise
    return 6985h
  if cod.internal_buffer is not formatted as described in Table 15-23
    return 6E09h
  if a certificate from chain does not comply with X.509 ASN.1 as per Listing A-4
    return 6400h
  if a certificate from chain cannot be verified as per verifier rules defined in Appendix A.4.7 and data defined in Listing 15-5,

```

[Listing 15-13](#), and [Listing 15-16](#)

```
15  return 6400h
16  if Endpoint certificate configuration has different vehicle_identifier than sender endpoint
17  return 6E10h
18  if Endpoint certificate any bit from option_group_1 is set while the same bit is cleared on sender endpoint
19  return 6E11h
20  if Endpoint certificate option_group_1 bit4 is set while bit5 is cleared on sender endpoint
21  return 6E12h
22  if Endpoint certificate option_group_2 bit0 or bit6 or bit7 is set while the same bit is
23  cleared on sender endpoint
24  return 6E13h
25  if Endpoint certificate option_group_2 bit1 is cleared while the same bit is set
26  on sender endpoint
27  return 6E14h
28  if Endpoint certificate configuration has different vehicle_PK than nvm.endpoint.vehicle_PK
29  return 6E15h
30  if Endpoint certificate contains authorized_PKs that are not part of nvm.endpoint.authorized_PK configuration
31  return 6E16h
32  if External CA certificate stored in cod.internal_buffer cannot be verified
33  with one of nvm.endpoint.authorized_PK public keys
34  return 6E19h
35  if Instance CA certificate stored in cod.internal_buffer cannot be verified with the public key of the External CA certificate
36  return 6E20h
37  if Endpoint certificate stored in cod.internal_buffer cannot be verified with the public key of the Instance CA certificate
38  return 6E21h
39  generate 8 bytes random as per Listing 15-40 to be added in data_fields according to Table 15-24
40  generate attestation signature over fields from Table 15-24 according to Listing 15-43 using nvm.endpoint.SK
41  if mailbox_offset and mailbox_length present in command payload
42  if Encryption key certificate not present in cod.internal_buffer
43  return 6E23h
44  if Encryption Key certificate present in cod.internal_buffer cannot be verified with
45  the public key of the Endpoint certificate present in cod.internal_buffer
46  return 6E24h
47  generate ephemeral key pair encsender_ePK, encsender_eSK according to Listing 15-41
48  generate KEseed according to Listing 15-45 using encsender_eSK, encreceiver_ePK
49  receiver_PK ← public key from receiver endpoint extracted from Endpoint certification present in internal buffer
50  derivation ← 08h
51  KEseed_half ← 16 most significant bytes of KEseed
52  generate KEenc according to Listing 15-49 using KEseed_half, encsender_ePK, encreceiver_ePK,
53  derivation, receiver_PK
54  derivation ← 12h
55  KEseed_half ← 16 least significant bytes of KEseed
56  generate KEmac according to Listing 15-49 using KEseed_half, encsender_ePK, encreceiver_ePK,
57  derivation, receiver_PK
58  encrypt and mac the selected confidential mailbox area according to Listing 15-50
59  if required by endpoint configuration
60  replace the exported portion of confidential mailbox with random bytes
61  clear cod.internal_buffer
62  fill cod.internal_buffer with signature, random[, encrypted_mailbox, mac, encsender_ePK] as described in Table 15-26
63  response_length ← length written in cod.internal_buffer
64  return response_length (2 bytes big-endian)
65  end
```

1 15.3.2.8 VIEW command

- 2 This command returns information on endpoints and Instance CA. The response is written in the
3 internal buffer and accessible using READ BUFFER command.

In case the internal buffer size does not allow storage of all requested data elements, the internal buffer is filled with the maximum number of data elements permitted. No truncated data elements are written in the internal buffer.

The applet should respond with a warning SW 6300_h (or any implementation-specific SW in the range 63XX_h or 62XX_h) to indicate that requested data is larger than the internal buffer size, and the length of the data written to the internal buffer which may be read using READ BUFFER command.

key_identifier is defined as the SHA-1 hash of the value of the BIT STRING subjectPublicKey of the target endpoint (excluding the tag, length, and number of unused bits).

command: CLA2 76 P1 P2 Lc (key_identifier) 00

response: [response_length] 9000

The CLA2 is as defined in [Table 15-3](#).

Table 15-27: View Parameters

P1	P2	Lc	Payload	Description	Domain Version
00 _h	00–FF _h	0	none	Request a list of LV-formatted key identifiers starting from the n-th element in memory, n represented by P2 (dummy endpoints are not returned).	Internal. N/A
01 _h	00 _h	20	key_identifier	Request endpoint setup information which includes all tags in the Table 15-54	Internal. N/A
02 _h	00 _h	20	key_identifier	Request endpoint creation certificate. Certificate parts (including certificate signature) may be re-generated.	Internal. N/A
03 _h	00 _h	0	none	Request current number of endpoints created in instance.	Internal. N/A
04 _h	00 _h	20	key_identifier	Request endpoint state active 01 _h /terminated 00 _h .	Internal. N/A
80 _h	00–FF _h	0	none	Request a list of LV-formatted instance CA identifiers starting from the n-th element in memory, n represented by P2.	Internal. N/A
82 _h	00 _h	0	none	Request instance additional information.	Internal. N/A
83 _h	00 _h	0	none	Request current number of Instance CA created in instance.	Internal. N/A
84 _h	00 _h	20	key_identifier	Request list of 1-byte transaction codes triggering user authentication on contactless interface; see Table 9-1 and Table 9-2 .	Internal. N/A
85 _h	00 _h	20	key_identifier	Request list of 1-byte transaction codes triggering user authentication on wired interface, see Table 9-1 and Table 9-2 .	Internal. N/A

Table 15-28: View Endpoint Identifier Response in Internal Buffer

Length	endpoint_identifier
20	key_identifier[n]
20	key_identifier[n+1]
20	...
20	key_identifier[last available Endpoint index (end of list) OR last Endpoint index that can fit in the internal buffer]

1

Table 15-29: View Instance CA Response in Internal Buffer

Length	instance_CA_identifier
1–30	instance_CA_identifier[n]
1–30	instance_CA_identifier[n+1]
1–30	...
1–30	instance_CA_identifier[last available Instance CA index (end of list) OR last Instance CA index that can fit in the internal buffer]

2

Listing 15-18: VIEW Processing

```

1  input: (key_identifier, instance_CA_identifier)
2  output: response_length
3  begin
6    if key_identifier or instance_CA_identifier provided as input does not exist
7      return 6400h
8    if P1=01h or P1=02h or P1=84h or P1=85h
9      if target endpoint with matching key_identifier is terminated
10     return 6A88h
11   if P1 = 01h
12     write in cod.internal_buffer endpoint setup information which includes all tags in Table 15-54, tag 4Eh shall be set to the
        current value of the key_slot
13   else if P1 = 02h
14     write in cod.internal_buffer endpoint creation certificate as per Listing 15-5
15   else if P1 = 03h
16     write in cod.internal_buffer current number of endpoints created (1 byte)
17   else if P1 = 04h
18     if endpoint designated by key_identifier is active
19       write in 01h in cod.internal_buffer (1 byte)
20     else
21       write 00h in cod.internal_buffer (1 byte)
22   else if P1 = 82h
23     write in cod.internal_buffer instance additional information formatted as per Table 15-4
24   else if P1 = 83h
25     write in cod.internal_buffer current number of Instance CA created (1 byte)
26   else if P1 = 84h
27     cod.internal_buffer <- nvm.endpoint.transaction_code_list_cless
28   else if P1 = 85h
29     cod.internal_buffer <- nvm.endpoint.transaction_code_list_wired
30   else if P1 = 00h
31     write in cod.internal_buffer list of key_identifier starting from n-th element as per Table 15-28 with n=P2
32   else if P1 = 80h
33     write in cod.internal_buffer list of instance_CA_identifier starting from n-th element as per Table 15-29 with n=P2
34     response_length = number of bytes written in cod.internal_buffer
35   if requested data bigger than internal buffer size
36     return response_length (2 bytes big-endian), 6300h (or implementation specific warning SW in the ranges of 63XXh or
        62XXh)
37   else
38     return response_length (2 bytes big-endian)
39   end

```

3 15.3.2.9 AUTH0 command

4 This command allows the vehicle to initiate the authentication procedure. In case a fast
5 transaction is requested by the vehicle, a cryptogram is returned, allowing the vehicle to
6 tentatively proceed with a fast transaction.

command: CLA3 80 P1 P2 Lc [Table 15-31] 00

response: [Table 15-32] 9000

The CLA3 is as defined in Table 15-3.

Table 15-30: AUTH0 P1, P2 Parameters

Description
P1 bits 0–7: bit0 Fast(1)/Standard(0) transaction request bit1 RFU (shall be set to 0) bit2 EXCHANGE commands will(1)/might(0) be sent during current transaction bit3–7 RFU (shall be set to 0)
P2 bits 0–7: bit0–7 domain specific transaction_code
Note: P1, P2 = FFFF _h is reserved (shall not be used in AUTH0)

Table 15-31: AUTH0 Command Payload

Tag	Length (bytes)	Description	Field is	Domain Version
5C _h	2	protocol_version, the Digital Key applet transaction (V-D-TX) protocol version (ver.high ver.low) selected by vehicle, shall be one of the versions proposed by DK Applet in SELECT response	mandatory	V-D-TX
87 _h	65	vehicle_ePK prepended by 04 _h	mandatory	V-D-TX
4C _h	16	transaction_identifier (randomly generated)	mandatory	V-D-TX
4D _h	8	vehicle_identifier (see Section 15.2)	mandatory	V-OD-FW

Table 15-32: AUTH0 Response Payload

Tag	Length (bytes)	Description	Field is	Domain Version
86 _h	65	endpoint_ePK, the applet generated ephemeral public key prepended by 04 _h	mandatory	V-D-TX
9D _h	16	cryptogram, the authentication cryptogram returned by the endpoint	conditional	V-D-TX

Cryptogram (Tag 9Dh) shall not be included in the response payload if a standard transaction is requested by the vehicle (AUTH0 with P1 bit0 = 0), otherwise it shall be present.

Listing 15-19: AUTH0 Processing

```
input: protocol_version, P1, P2, vehicle_ePK, transaction_identifier, vehicle_identifier
output: endpoint_ePK, (cryptogram)
begin
  if cod.transaction_state != select_done
    return 6400h
  if protocol_version is not supported (i.e., not part of the nvm.supported_protocol_versions_tlv)
    return 6400h
  cod.current_protocol_version ← protocol_version
  generate an ephemeral key pair cod.endpoint_ePK, cod.endpoint_eSK according to Listing 15-41
  endpoint_ePK ← 04h || cod.endpoint_ePK
  cod.flag ← P1 || P2
```



```
12 begin constant time or unbiased random time section
13 find endpoint for corresponding vehicle_identifier excluding endpoints terminated or with visibility disabled
14 if no endpoint found
15     endpoint ← dummy_endpoint
16 if fast transaction requested
17     send to digital key framework notify_endpoint_not_found_fast with P1, P2
18 else
19     send to digital key framework notify_endpoint_not_found_standard with P1, P2
20 if fast transaction requested but not allowed on current interface as per Table 15-13
21     endpoint ← dummy_endpoint
22 end constant time or unbiased random time section
23 if interface is contactless
24     if transaction_code from P2 is present in nvm.endpoint.transaction_code_list_cless
25         if user authentication not performed
26             send to digital key framework notify_user_authentication_not_performed with P1, P2
27             endpoint ← dummy_endpoint
28 else
29     if transaction_code from P2 is present in nvm.endpoint.transaction_code_list_wired
30         if user authentication is not performed
31             send to digital key framework notify_user_authentication_not_performed with P1, P2
32             endpoint ← dummy_endpoint
33 cod.vehicle_ePK ← vehicle_ePK
34 cod.transaction_identifier ← transaction_identifier
35 cod.mac_chaining ← 00000000000000000000000000000000h
36 cod.counter ← 00h
37 cod.endpoint ← endpoint
38 if Standard transaction requested
39     cod.transaction_state ← auth0_std_done
40     send to digital key framework notify_standard with nvm.endpoint.identifier with P1, P2
41     return endpoint_ePK (65 bytes)
42 else
43     cod.transaction_state ← auth0_fast_done
44     if endpoint.kpersistent_established = true
45         send to digital key framework
46         notify_fast_kpersistent_established with nvm.endpoint.identifier with P1, P2
47     else
48         send to digital key framework
49         notify_fast_kpersistent_not_established with nvm.endpoint.identifier with P1, P2
50 // NOTE: The cryptogram computation below applies
51 // even if endpoint.kpersistent_established is set to false,
52 // as nvm.endpoint.Kpersistent was initialized to a random value at endpoint creation
53
54 interface_byte ← C3h for wired interface or 5Eh for contactless interface
55 info ← cod.vehicle_ePK.x || cod.endpoint_ePK.x || cod.transaction_identifier || interface_byte ||
56     cod.flag || "VolatileFast" || || 5Ch || 02h || cod.current_protocol_version
57 keying_material_length ← 64
58 compute derived_keys according to Listing 15-46 using nvm.endpoint.Kpersistent, info, keying_material_length
59 KCmac ← subset of derived_keys at offset 0 with length 16
60 cod.Kenc ← subset of derived_keys at offset 16 with length 16
61 cod.Kmac ← subset of derived_keys at offset 32 with length 16
62 cod.Krmac ← subset of derived_keys at offset 48 with length 16
63
64 compute cryptogram according to Listing 15-51 using KCmac
65 return endpoint_ePK (65 bytes), cryptogram (16 bytes)
66 end
```

15.3.2.10 AUTH1 command

This command allows mutual authentication and establishment of a secure channel between the vehicle and device.

This command may be used to optionally pre-compute a ranging key. If this option is implemented, this command shall produce 80 bytes of keying material and the computed ranging key shall subsequently be made available upon reception of the CREATE RANGING KEY command. Otherwise, this command shall produce 48 bytes of keying material only.

command: CLA3 81 00 00 Lc [Table 15-34] 00

response: [encrypted_payload] [mac] 9000

The CLA3 is as defined in Table 15-3.

Table 15-33: AUTH1 Vehicle Authentication Data Fields

Tag	Length (bytes)	Description	Field is	Domain Version
4D _h	8	vehicle_identifier	mandatory	V-OD-FW
86 _h	32	endpoint_ePK.x	mandatory	V-D-TX
87 _h	32	vehicle_ePK.x	mandatory	V-D-TX
4C _h	16	transaction_identifier	mandatory	V-D-TX
93 _h	4	usage = 415D9569 _h	mandatory	V-D-TX

Table 15-34: AUTH1 Command Payload

Tag	Length (bytes)	Description	Field is	Domain Version
9E _h	64	vehicle_sig computed as per Listing 15-43 using fields from Table 15-33 and vehicle_SK	mandatory	V-D-TX

Table 15-35: AUTH1 Response Payload Before Encryption

Tag	Length (bytes)	Description	Field is	Domain Version
4E _h	1–8	key_slot	mandatory	V-OD-FW
9E _h	64	endpoint_sig, computed as per Listing 15-43 using fields from Table 15-36 and endpoint_SK	mandatory	V-D-TX
4A _h	variable	confidential_mailbox_data_subset. This tag shall be included if it was included previously in SETUP ENDPOINT command	conditional	V-OD-FW
4B _h	variable	private_mailbox_data_subset. This tag shall be included if it was included previously in SETUP ENDPOINT command	conditional	V-OD-FW

Table 15-36: Endpoint Authentication Data Fields

Tag	Length (bytes)	Description	Field is	Domain Version
4D _h	8	vehicle_identifier	mandatory	V-OD-FW
86 _h	32	endpoint_ePK.x	mandatory	V-D-TX
87 _h	32	vehicle_ePK.x	mandatory	N/A
4C _h	16	transaction_identifier	mandatory	V-OD-FW

Tag	Length (bytes)	Description	Field is	Domain Version
93 _h	4	usage = 4E887B4C _h	mandatory	V-OD-FW

1 *Listing 15-20: AUTH1 Processing*

```

1  input: vehicle_sig
2  output: encrypted_payload, mac
3  begin
4    if cod.transaction_state != auth0_std_done AND cod.transaction_state != auth0_fast_done
5      return 6400h
6    if standard transaction not allowed on this interface as per Table 15-13
7      return 6900h
8    endpoint ← cod.endpoint
9    fetch nvm.endpoint.vehicle_PK
10   verify vehicle_sig according to Listing 15-44 using nvm.endpoint.vehicle_PK and fields from Table 15-33
11   if vehicle_sig is not verified
12     send to digital key framework notify_vehicle_authentication_failed
13     return 6400h
14   send to digital key framework notify_vehicle_authentication_success
15   compute cod.Kdh according to Listing 15-45 using cod.transaction_identifier, cod.vehicle_ePK, cod.endpoint_eSK
16   interface_byte ← C3h for wired interface or 5Eh for contactless interface
17   info ← cod.vehicle_ePK.x || cod.endpoint_ePK.x || cod.transaction_identifier || interface_byte ||
18     cod.flag || "Volatile" || 5Ch || 02h || cod.current_protocol_version
19   if wired interface
20     keying_material_length ← 48 or 80
21   else
22     keying_material_length ← 48
23   compute derived_keys according to Listing 15-46 using cod.Kdh, info, keying_material_length
24   cod.Kenc ← subset of derived_keys at offset 0 with length 16
25   cod.Kmac ← subset of derived_keys at offset 16 with length 16
26   cod.Krmac ← subset of derived_keys at offset 32 with length 16
27   if wired interface and keying_material_length = 80
28     cod.URSK ← subset of derived_keys at offset 48 with length 32
29   info ← cod.vehicle_ePK.x || cod.endpoint_ePK.x || cod.transaction_identifier || interface_byte ||
30     cod.flag || "Persistent" || 5Ch || 02h || cod.current_protocol_version
31   keying_material_length ← 32
32   compute derived_keys according to Listing 15-46 using cod.Kdh, info, keying_material_length
33   Kpersistent ← subset of derived_keys at offset 0 with length 32
34   append nvm.endpoint.key_slot to response payload
35   atomic start
36     if fast transaction allowed by endpoint configuration as per Table 15-13 (option_group_1 bit1 and/or bit3 is set)
37       nvmmwrite nvm.endpoint.Kpersistent ← Kpersistent
38       nvmmwrite nvm.endpoint.kpersistent_established = true
39   atomic commit
40   compute endpoint_sig over fields Table 15-36 according to Listing 15-43 using endpoint.SK
41   append endpoint_sig to response payload
42   append confidential_mailbox_data_subset portion to response payload as per configuration in Section 15.3.2.21
43   append private_mailbox_data_subset to response payload
44   encrypt and mac response payload according to Listing 15-48 using cod.Kenc, cod.Krmac, cod.counter, cod.mac_chaining
45   cod.transaction_state ← auth1_done
46   return mac, encrypted_payload
47 end

```

2 *15.3.2.11 PRESENCE0 command*

3 This command allows the vehicle to initiate the Check Presence transaction.

1 **command:** CLA3 3D 00 00 Lc [Table 15-31] 00

2 **response:** [Table 15-37] 9000

3 The CLA3 is as defined in Table 15-3.

4 *Table 15-37: PRESENCE0 Response Payload*

Tag	Length (bytes)	Description	Field is	Domain Version
86 _h	65	endpoint_ePK, the applet generated ephemeral public key prepended by 04 _h	mandatory	V-D-TX

5 *Listing 15-21: PRESENCE0 Processing*

```
1 input: protocol_version, vehicle_ePK, transaction_identifier, vehicle_identifier
2 output: endpoint_ePK
3 begin
4   if cod.transaction_state != select_done
5     return 6400h
6   if protocol_version is not supported (i.e., not part of nvm.supported_protocol_versions_tlv)
7     return 6400h
8   cod.current_protocol_version ← protocol_version
9   generate an ephemeral key pair cod.endpoint_ePK, cod.endpoint_eSK according to Listing 15-41
10  endpoint_ePK ← 04h || cod.endpoint_ePK
11  cod.flag ← 5Bh
12  begin constant time or unbiased random time section
13    find endpoint for corresponding vehicle_identifier excluding endpoints terminated or with visibility disabled
14    if no endpoint found
15      endpoint ← dummy_endpoint
16      send to digital key framework notify_endpoint_not_found_presence
17    end constant time or unbiased random time section
18    cod.endpoint ← endpoint
19    cod.vehicle_ePK ← vehicle_ePK
20    cod.transaction_identifier ← transaction_identifier
21    cod.transaction_state ← presence0_done
22    cod.mac_chaining ← 00000000000000000000000000000000h
23    cod.counter ← 00h
24    send to digital key framework notify_presence with nvm.endpoint.identifier
25    return endpoint_ePK (65 bytes)
26 end
```

6 15.3.2.12 *PRESENCE1 command*

7 This command allows an authenticated vehicle to retrieve the endpoint key_slot.

8 **command:** CLA3 3E 00 00 Lc [Table 15-38] 00

9 **response:** [encrypted_payload] [mac] 9000

10 The CLA3 is as defined in Table 15-3.

11 *Table 15-38: PRESENCE1 Command Payload*

Tag	Length (bytes)	Description	Field is	Domain Version
9E _h	64	vehicle_sig computed as per Listing 15-43 using fields from Table 15-39 and vehicle_SK	mandatory	V-D-TX

1 *Table 15-39: PRESENCE1 Vehicle Authentication Data Fields*

Tag	Length (bytes)	Description	Field is	Domain Version
4D _h	8	vehicle_identifier	mandatory	V-OD-FW
86 _h	32	endpoint_ePK.x	mandatory	V-D-TX
87 _h	32	vehicle_ePK.x	mandatory	V-D-TX
4C _h	16	transaction_identifier	mandatory	V-D-TX
93 _h	4	usage = 47165979 _h	mandatory	V-D-TX

2 *Table 15-40: PRESENCE1 Response Payload Before Encryption*

Tag	Length (bytes)	Description	Field is	Domain Version
4E _h	1–8	key_slot	mandatory	V-OD-FW

3 *Listing 15-22: PRESENCE1 Processing*

```

1  input: vehicle_sig
2  output: encrypted_payload, mac
3  begin
4    if cod.transaction_state != presence0_done
5      return 6400h
6    cod.transaction_state ← cp_done
7    endpoint ← cod.endpointh
8    verify vehicle_sig according to Listing 15-44 using nvm.endpoint.vehicle_PK and fields from Table 15-39
9    if vehicle_sig is not verified
10     send to digital key framework notify_vehicle_authentication_failed
11     return 6400h
12    send to digital key framework notify_vehicle_authentication_success
13    compute Kdh according to Listing 15-45 using cod.transaction_identifier, cod.vehicle_ePK, cod.endpoint_eSK
14    interface_byte ← 5Eh for contactless interface
15    info ← cod.vehicle_ePK.x || cod.endpoint_ePK.x || cod.transaction_identifier || interface_byte ||
16      cod.flag || "Volatile" || 5Ch || 02h || cod.current_protocol_version
17    keying_material_length ← 48
18    compute derived_keys according to Listing 15-46 using Kdh, info, keying_material_length
19    cod.Kenc ← subset of derived_keys at offset 0 with length 16
20    cod.Krmac ← subset of derived_keys at offset 32 with length 16
21
22    append nvm.endpoint.key_slot to response payload as per Table 15-40
23    encrypt and mac response payload according to Listing 15-48 using cod.Kenc, cod.Krmac, cod.counter, cod.mac_chaining
24    return mac, encrypted_payload
25  end

```

4 15.3.2.13 READ BUFFER command

5 This command is used to read temporarily stored data in an internal buffer. This supplements the
6 APDU buffer for large payloads. The following command formats are allowed:

- 7 • If no secure channel is to be established between the Digital Key framework and the
8 Digital Key applet, then either command format 1 or command format 2 may be used and
9 implemented by the applet.
- 10 • If a secure channel is to be established between the Digital Key framework and the
11 Digital Key applet (option A), command format 2 shall be used and shall be implemented
12 by the applet.

Command format 1:

command: CLA2 B0 [offsetmsb] [offsetlsb] Le

response: [response] 90 00

The CLA2 is as defined in Table 15-3.

Command format 2:

command: CLA2 B0 [offsetmsb] [offsetlsb] Lc [Table 15-41] 00

response: [response] 90 00

The CLA2 is as defined in Table 15-3.

Table 15-41: READ BUFFER Command Payload for Format 2

Tag	Length (bytes)	Description	Field is	Domain Version
80 _h	1	1-byte unsigned size of the data that shall be read from the internal buffer	mandatory	Internal. N/A

Listing 15-23: READ BUFFER Processing for Command Format 1

```
input: offsetmsb, offsetlsb, Le
output: response
begin
  if data to be read overflows the cod.internal_buffer size
    return 6400h
  read cod.internal_buffer at offsetmsb || offsetlsb
  return response
end
```

Listing 15-24: READ BUFFER Processing for Command Format 2

```
input: offsetmsb, offsetlsb, size_of_data
output: response
Begin
  if size_of_data != 00h
    if size_of_data > remaining size of available data at offset offsetmsb || offsetlsb or size_of_data > maximum response size allowed
      return 6400h or 6700h or 6985h
    return response of size_of_data bytes at offset offsetmsb || offsetlsb from the cod.internal_buffer
  end
```

15.3.2.14 WRITE BUFFER command

This command is used to temporarily store data in an internal RAM buffer. This supplements the APDU buffer for large payloads.

command: CLA2 D0 [offsetmsb] [offsetlsb] Lc [data]

response: 90 00

The CLA2 is as defined in Table 15-3.

Listing 15-25: WRITE BUFFER Processing

```
input: offsetmsb, offsetlsb, data
output: n/a
begin
  if data to be be written overflows the cod.internal_buffer size
    return 6400h
  write cod.internal_buffer at offsetmsb || offsetlsb with data from command payload
```

7 **return**
8 **end**

15.3.2.15 EXCHANGE command

This command reads, writes or sets data from confidential/private mailboxes. This command is available after AUTH0 and/or AUTH1, depending on endpoint creation parameters. The commands and responses are always wrapped in the currently established secure channel.

The command also is used to send notification messages to the Digital Key framework. In such cases, the messages are not stored in mailbox.

In case command MAC verification fails, the applet returns 6982_h, and the atomic session, if any, is aborted.

The set request fills the indicated memory area with the provided single byte value.

Multiple read, write, set, and notify operations can be present in a single EXCHANGE command. Device shall support up to one notify operation to be transferred per EXCHANGE command, if more than one notify operation is sent in an EXCHANGE command the behavior is undefined

All read operations contained in an EXCHANGE command are returned in the order they appear in the request buffer. All write/set operations contained in an EXCHANGE command are written atomically and in the order they appear in the request buffer if the Atomic Session indicator flag is set to 0. Write/set operations are atomic per group of EXCHANGE commands processed as part of an atomic session.

An atomic session spanned over multiple commands is executed by setting the Atomic Session indicator flag to 1 on a series of consecutive EXCHANGE commands. The session is closed, and all data written in NVM by setting the Atomic Session indicator flag to 0 on the last EXCHANGE command of the series.

When multiple EXCHANGE commands are part of an atomic session, all written values are effectively readable only after successful execution of a command with the Atomic Session indicator flag set to 0. All read operations occurring before that point return the old value.

If the transaction has been started over the wired interface, as conditionally permitted depending on endpoint configuration, the Digital Key framework shall avoid prematurely interrupting the transaction, e.g., by reselecting the applet. However, if it hasn't been notified of transaction end after some unusually long time period (i.e., timeout), the Digital Key framework may decide to forcefully end the transaction, e.g., by reselecting the applet.

command: CLA4 C9 00 00 Lc [encrypted_command_payload] [command_mac] 00

response: [encrypted_response_payload] [response_mac] 90 00

The CLA4 is as defined in Table 15-3.

Table 15-42: Exchange Command Decrypted Payload

Tag	Length (bytes)	Description	Field is	Domain Version
option byte: bit0	Atomic Session indicator start(1)/stop(0), other bits RFU (left to 0)		mandatory	V-D-TX
88 _h	3	offsetmsb offsetlsb length, read request in private mailbox	optional	V-D-TX
89 _h	3	offsetmsb offsetlsb length, read request in confidential mailbox	optional	V-D-TX

Tag	Length (bytes)	Description	Field is	Domain Version
8A _h	variable	offsetmsb offsetlsb data, write request in private mailbox	optional	V-D-TX
8B _h	variable	offsetmsb offsetlsb data, write request in confidential mailbox	optional	V-D-TX
8C _h	4	offsetmsb offsetlsb length value, set request in private mailbox	optional	V-D-TX
8D _h	4	offsetmsb offsetlsb length value, set request in confidential mailbox	optional	V-D-TX
95 _h	5	offsetmsb offsetlsb lengthmsb lengthlsb value, set request in private mailbox	optional	V-D-TX
96 _h	5	offsetmsb offsetlsb lengthmsb lengthlsb value, set request in confidential mailbox	optional	V-D-TX
8E _h	variable	notify Digital Key framework with data present in this field. Data present in this field is stored in the Digital Key framework to be transmitted to the Vehicle OEM app. The maximum size supported for the value field of this TLV object is 211 bytes (size available for the <i>notify_message_in_exchange</i> notification to the Digital Key framework). This is reduced by 40 bytes if the EXCHANGE command also includes one or more write or set requests (to manage the <i>notify_mailbox_written</i> notification). In any case, the size of this value field is limited by the available payload in the EXCHANGE command.	optional	V-D-TX
...other read, write or set requests			optional	

1 *Table 15-43: Exchange Response Decrypted Payload*

Length (bytes)	Description	Field is	Domain Version
variable	data read from request 1	conditional	V-D-TX
variable	data read from request 2	conditional	V-D-TX
...data read from request n		conditional	

2 *Listing 15-26: EXCHANGE Processing*

```

1  input: encrypted_command_payload, command_mac
2  output: encrypted_response_payload, response_mac
3  begin
4    if cod.counter = FFn
5      cod.transaction_state ← select_done
6      cod.atomic_session ← stopped
7      return 6900h
8
9    if cod.transaction_state != exchange_done and cod.transaction_state != auth1_done
      and cod.transaction_state != auth0_fast_done
10     return 6400h
11
12    if cod.transaction_state == auth0_fast_done and EXCHANGE not allowed after AUTH0
13     return 6400h
14

```



```
15   cod.counter ← cod.counter + 1
16   endpoint ← cod.endpointi
17   verify and decrypt command according to Listing 15-47
18   inputs: cod.Kenc, cod.Kmac, cod.counter, cod.mac_chaining
19   output: decrypted_command_payload, mac_chaining_out
20   if command mac verification fails
21     cod.transaction_state ← select_done
22     cod.atomic_session ← stopped
23     return 6982h
24   cod.mac_chaining ← mac_chaining_out
25   cod.transaction_state ← exchange_done
26   if total requested output data in decrypted_command_payload is more than 239 bytes
27     return 6400h
28   if write/read/set requests present in decrypted_command_payload are out of mailbox boundaries
29     return 6400h
30   if notification requests are present in command payload
31     send to digital key framework notify_message_in_exchange
32   execute all notification requests from Table 15-42
33   execute all read requests from Table 15-42 on endpoint mailboxes
34   append read data to response payload for all read requests as per Table 15-43 from endpoint mailboxes
35   if Atomic Session indicator bit = 1
36     cod.atomic_session ← started
37   if cod.atomic_session = started
38     if all write/set requests exceeds cod.commit_buffer size
39       cod.atomic_session ← stopped
40       return 6A84h
41   store all write/set requests from Table 15-42 in cod.commit_buffer
42   if Atomic Session indicator bit = 0
43     atomic start
44       if cod.commit_buffer is not empty
45         execute all write/set requests pending in cod.commit_buffer on endpoint mailboxes
46         send to digital key framework notify_mailbox_written
47     atomic commit
48     cod.atomic_session ← stopped
49   else
50     atomic start
51     if decrypted_command_payload contains write or set requests
52       execute all write/set requests from Table 15-42 on endpoint mailboxes
53       send to digital key framework notify_mailbox_written
54     atomic commit
55   compute and return encrypted_response_payload and response_mac according to
56   Listing 15-48 using cod.Kenc, cod.Krmac, cod.mac_chaining, cod.counter
57   end
```

1 15.3.2.16 *CONTROL FLOW* command

2 This command allows the vehicle to indicate the final success or failure of the transaction or to
3 signal application-specific codes.

4 The P2 field is used for vehicle error codes, domain specific codes, or timing information.

5 The vehicle error codes are hints provided by the vehicle to the device in case of a transaction
6 failure. The decision to send such error codes is vehicle implementation-specific.

7 The variables P1 and P2 are version controlled under the V-D-TX domain version.

8 **command:** CLA3 3C [Table 15-44] [Table 15-45]

9 **response:** 90 00

1 The CLA3 is as defined in Table 15-3.

2 *Table 15-44: CONTROL FLOW P1 Parameters*

P1 value	Description
00 _h	transaction finished with failure
01 _h	transaction finished with success
40 _h	application specific
other values	RFU

3 *Table 15-45: CONTROL FLOW P2 Parameters*

P2 value	Description
00 _h	with P1 = 00 _h , 01 _h , or 40 _h , no information provided
00 _h –7F _h	Only with P1 = 00 _h , this range is used for the following vehicle error codes: 00 _h no information provided 01 _h public key not found 02 _h public key expired 03 _h public key not trusted 04 _h invalid signature 05 _h invalid channel 06 _h invalid data format 07 _h invalid data content 08 _h mailbox version unknown 09 _h attestation package verification failed 0A _h –7F _h RFU
80 _h –FF _h	With P1 = 00 _h , 01 _h , or 40 _h , this valid range is shown in Table 15-46

4 *Table 15-46: CONTROL FLOW P1/P2 Values for Applet*

P1	P2	Description	Reference
00 _h	A0 _h	Key deleted in/not known to vehicle. Vehicle was offline during key deletion in vehicle or unknown key without attestation package detected	See Note 4
00 _h	A8 _h	Digital Key sharing disabled, cannot accept the Digital Key right now	Section 12.4.6
00 _h	B0 _h	Cannot lock the vehicle because not all doors/trunk are closed	See Note 4, 5
00 _h	B1 _h	Cannot lock the vehicle because it is not in parking state (e.g. engine is still running/vehicle is still in driving state)	See Note 4, 5
01 _h	B0 _h	First approach successful	Section 19.5.8.2
01 _h	81 _h	Successful end, vehicle has completed owner pairing phase 3	Section 6.3.4.2
01 _h	90 _h	End, key is tracked, and all data have been written successfully into the mailboxes	Section 6.3.5.4
01 _h	91 _h	End, key is not tracked, key sharing is not possible, and the owner needs to go online to track the key before using it	Section 6.3.5.4

P1	P2	Description	Reference
40 _h	81 _h	Optional token refill start	Section 6.3.5.2
40 _h	82 _h	Continue, device key Attestation package delete start	Section 6.3.5.3
40 _h	83 _h	Optional service data write start	See Note 1
40 _h	88 _h	Continue, key tracking response received in device, next step is to read the receipt from the mailbox	Section 6.3.5.2
40 _h	89 _h	Continue, key tracking response received in car, go directly to verification of KTS signature	Section 6.3.5.2
40 _h	90 _h	Extend reader processing time (do nothing on device)	See Note 2
40 _h	A0 _h	Long processing time (UI indication)	See Note 3
<i>Note 1:</i> Vehicle OEM may use this option for providing application-specific service data after engine start sequence. <i>Note 2:</i> Extended reader processing time is a command that can be sent by the reader to prevent a reader-side timeout. It has no effect on the device. <i>Note 3:</i> When the vehicle executes operations that take longer than the usual UI-tolerated times, it should send this indication to the device. The device can show an appropriate UI, which is out of scope of this specification. <i>Note 4:</i> Used to indicate unsuccessful completion of transaction or inability to perform intended action due to a conflicting state on the vehicle. <i>Note 5:</i> Typically, only relevant for NFC transactions.			

1

Listing 15-27: CONTROL FLOW Processing

```

1  input: P1, P2
2  output: none
3  begin
4    if cod.transaction_state = auth0_std_done or cod.transaction_state = auth1_done or cod.transaction_state =
      auth0_fast_done or cod.transaction_state = exchange_done or cod.transaction_state = create_rk_done
5      if P1 == 00h
6        cod.transaction_state ← select_done
7        cod.atomic_session ← stopped
8        send to digital key framework notify_end_of_transaction with P1, P2 of CONTROL FLOW, P1, P2 of AUTH0 command
9      else if P1 == 01h
10        cod.transaction_state ← select_done
11        cod.atomic_session ← stopped
12        send to digital key framework notify_end_of_transaction with P1, P2 of CONTROL FLOW, P1, P2 of AUTH0 command
13      else
14        send to digital key framework notify_application_specific with P1, P2 of CONTROL FLOW, P1, P2 of AUTH0 command
15    else
16      if P1 == 00h or P1 == 01h
17        send to digital key framework notify_end_of_transaction with P1, P2 of CONTROL FLOW, outside of a fast or standard
          transaction
18      else
19        send to digital key framework notify_application_specific with P1, P2 of CONTROL FLOW, outside of a fast or standard
          transaction
20  End

```

2 15.3.2.17 CREATE ENCRYPTION KEY command

3 Create a confidential mailbox encryption key attestation in order to allow data insertion in the
4 endpoint. This encryption key is stored in NVM and valid for only one usage with the SET

CONFIDENTIAL DATA command. After single usage with the SET CONFIDENTIAL DATA command, the key is erased from internal memory.

The attestation containing the encryption public key is stored in the internal buffer, accessible with the READ BUFFER command.

command: CLA2 8A 00 00 Lc [Table 15-47] 00

response: [response_length] 90 00

Table 15-47: CREATE ENCRYPTION KEY Command Payload

Tag	Length (bytes)	Description	Field is	Domain Version
50 _h	20	key_identifier, SHA-1 hash of the value of the BIT STRING subjectPublicKey of the target endpoint (excluding the tag, length, and number of unused bits)	mandatory	V-OD-FW

Table 15-48: Encryption Key Attestation Data Fields

Tag	Length (bytes)	Description	Field is	Domain Version
41 _h	1	version = 01 _h , version of Encryption Key Attestation	mandatory	N/A. Informative Only
92 _h	8	random	mandatory	N/A. Informative Only
5F49 _h	65	encreceiver_ePK, the encryption ephemeral public key prepended by 04 _h	mandatory	N/A. Informative Only
5D _h	20	authority_key_identifier, 160-bit SHA-1 hash of the value of the BIT STRING subjectPublicKey from the issuer (endpoint) certificate (as defined in Listing 15-5 excluding the tag, length, and number of unused bits)	mandatory	N/A. Informative Only
93 _h	4	usage = 49A8A741 _h	mandatory	N/A. Informative Only

Table 15-49: Encryption Key Attestation

Tag	Length (bytes)	Description	Field is	Domain Version
7F26 _h	variable	Encryption Key Attestation	mandatory	N/A. Informative Only
content of Table 15-48			mandatory	N/A. Informative Only
9E _h	64	Receiver endpoint signature over fields from Table 15-48 with endpoint.SK	mandatory	N/A. Informative Only

Listing 15-28: CREATE ENCRYPTION KEY Processing

```

input: key_identifier
output: response_length, encryption_key_attestation
begin
  if key_identifier does not match with any of the existing endpoints or target endpoint is terminated
    return 6A88h
  generate a key pair encreceiver_ePK, encreceiver_eSK according to Listing 15-41
  atomic start
    store nvm.endpoint.encreceiver.eSK using encreceiver_eSK
    nvm.endpoint.encreceiver.isActive ← true
  atomic commit
  generate signature using nvm.endpoint.SK and fields from Table 15-48 according to Listing 15-43
  generate data field encryption_key_attestation as per Table 15-49

```

```

13  store encryption_key_attestation in cod.internal_buffer
14  response_length ← length written in cod.internal_buffer
15  return response_length (2 bytes big-endian)
16  end

```

1 15.3.2.18 GET PRIVATE DATA command

2 Retrieve data in the private mailbox. The following command formats are allowed:

- 3 • If no secure channel is to be established between the Digital Key framework and the
- 4 Digital Key applet, then either Command Format 1 or Command Format 2 may be used
- 5 and implemented by the applet.
- 6 • If a secure channel is to be established between the Digital Key framework and the
- 7 Digital Key applet (option A), command format 2 shall be used and shall be implemented
- 8 by the applet.

9 Command Format 1:

10 **command:** CLA2 78 [offsetmsb] [offsetlsb] Lc [Table 15-50] Le

11 **response:** [response] 90 00

12 The CLA2 is as defined in Table 15-3.

13 Command Format 2:

14 **command:** CLA2 78 [offsetmsb] [offsetlsb] Lc [Table 15-50] 00

15 **response:** [response] 90 00

16 The CLA2 is as defined in Table 15-3.

17 Table 15-50: GET PRIVATE DATA Command Payload

Tag	Length (bytes)	Description	Field is	Domain Version
50 _h	20	key_identifier, SHA-1 hash of the value of the BIT STRING subjectPublicKey of the target endpoint (excluding the tag, length, and number of unused bits)	mandatory	V-OD-FW
80 _h	1	1-byte unsigned size of the data that shall be read from the internal buffer	mandatory for command format 2 not required for command format 1	V-OD-FW

18 Listing 15-29: GET PRIVATE DATA Processing for Command Format 1

```

1  input: key_identifier, offset_msb, offset_lsb, Le
2  output: response
3  begin
4    if key_identifier does not match with any existing endpoints or target endpoint is terminated
5      return 6A88h
6    if Le != 00h
7      if Le > remaining size of available data at offset offsetmsb || offsetlsb or Le > maximum response size allowed
8        return 6400h
9    else
10     if remaining size of available data at offset offsetmsb || offsetlsb > maximum response size allowed
11       return response of maximum response size allowed at offset offsetmsb || offsetlsb from the private mailbox
12     else
13       return response of remaining size of available data at offset offsetmsb || offsetlsb from the private mailbox

```

```

14  return response of Le bytes at offset offsetmsb || offsetlsb from the private mailbox
15  end

```

1 *Listing 15-30: GET PRIVATE DATA Processing for Command Format 2*

```

1  input: key_identifier, offset_msb, offset_lsb, size_of_data
2  output: response
3  begin
4    if key_identifier does not match with any existing endpoints or target endpoint is terminated
5      return 6A88h
6    if size_of_data != 00h
7      if size_of_data > remaining size of available data at offset offsetmsb || offsetlsb or size_of_data > maximum response
        size allowed
8      return 6400h or 6700h or 6985h
9    return response of size_of_data bytes at offset offset_msb || offset_lsb from the private mailbox
10 end

```

2 15.3.2.19 SET PRIVATE DATA command

3 Store data in the private mailbox.

4 **command:** CLA2 7A [offsetmsb] [offsetlsb] Lc [Table 15-51]

5 **response:** 90 00

6 The CLA2 is as defined in Table 15-3.

7 *Table 15-51: SET PRIVATE DATA Command Payload*

Tag	Length (bytes)	Description	Field is	Domain Version
50 _h	20	key_identifier, SHA-1 hash of the value of the BIT STRING subjectPublicKey of the target endpoint (excluding the tag, length, and number of unused bits)	mandatory	V-OD-FW
4B _h	variable	data, field to be written in private mailbox	mandatory	V-OD-FW

8 *Listing 15-31: SET PRIVATE DATA Processing*

```

1  input: key_identifier, offsetmsb, offsetlsb, data
2  output: none
3  begin
4    if key_identifier does not match with any existing endpoints or target endpoint is terminated
5      return 6A88h
6    atomic start
7      write command data in NVM private mailbox at offsetmsb || offsetlsb
8    atomic commit
9    return
10 end

```

9 15.3.2.20 SET CONFIDENTIAL DATA command

10 Store data in the confidential mailbox. The appropriate data fields as described Table 15-53, shall
11 have been previously loaded in the internal buffer at offset 0 using the WRITE BUFFER
12 command. The CREATE ENCRYPTION KEY shall have been executed for this endpoint, and
13 the encryption key shall still be active.

14 **command:** CLA2 7C [offsetmsb] [offsetlsb] Lc [Table 15-52]

15 **response:** 90 00

1 *Table 15-52: SET CONFIDENTIAL DATA Command Payload*

Tag	Length (bytes)	Description	Field is	Domain Version
50 _h	20	key_identifier, SHA-1 hash of the value of the BIT STRING subjectPublicKey of the target endpoint (excluding the tag, length, and number of unused bits)	mandatory	V-OD-FW

2 *Table 15-53: SET CONFIDENTIAL DATA Internal Buffer Content Before Processing*

Tag	Length (bytes)	Description	Field is	Domain Version
97 _h	65	encsender_ePK	mandatory	N/A. Informative Only
4A _h	variable	encrypted_mailbox mac	mandatory	N/A. Informative Only

3 *Listing 15-32: SET CONFIDENTIAL DATA Processing*

```

1  input: key_identifier, offsetmsb, offsetlsb
2  output:
3  begin
4    if key_identifier does not match with any existing endpoints or target endpoint is terminated
5      return 6A88h
6    if command is not allowed as per Table 15-13
7      return 6900h
8    if nvm.endpoint.encreceiver.isActive ← false
9      return 6400h
10   if cod.internal_buffer does not contain data described in Table 15-53
11     return 6400h
12   nvm.endpoint.encreceiver.isActive ← false
13   generate KEseed according to Listing 15-45 using nvm.endpoint.encreceiver.eSK, encsender_ePK
14   derivation ← 08h
15   KEseed_half ← 16 most significant bytes of KEseed
16   generate KEenc according to Listing 15-49 using KEseed_half, encsender_ePK, nvm.endpoint.encreceiver_ePK,
17     derivation, nvm.endpoint.PK
18   derivation ← 12h
19   KEseed_half ← 16 least significant bytes of KEseed
20   generate KEmac according to Listing 15-49 using KEseed_half, encsender_ePK, nvm.endpoint.encreceiver_ePK,
21     derivation, nvm.endpoint.PK
22   verify and decrypt the selected confidential mailbox area according to Listing 15-50
23   if verification fails
24     randomize nvm.endpoint.encreceiver.eSK
25     return 6982h
26   atomic start
27     randomize nvm.endpoint.encreceiver.eSK
28     write command data in NVM confidential mailbox at offsetmsb || offsetlsb
29   atomic commit
30   clear cod.internal_buffer
31   return
32 end

```

4 15.3.2.21 *SETUP ENDPOINT command*

5 This command is used to change the default private/confidential mailbox content returned in
6 AUTH1 response; by default, no mailbox content is returned in the AUTH1 response.

This command is used to enable/disable individual endpoint visibility. When visibility is disabled, the endpoint cannot be found during the AUTH0 command. Consequently, no transaction can be executed with the endpoint. Other management commands remain available. A newly created endpoint is visible by default. Persistent configurations are applied immediately upon request (overriding current volatile configuration, if any) and are reapplied on each SE power cycle. Volatile configurations are applied immediately upon request. When persistent and volatile configurations are present in the same command, the configurations are applied in the order mentioned in [Listing 15-33](#).

command: CLA2 7E 00 00 Lc [Table 15-54]
response: 90 00

The CLA2 is as defined in Table 15-3.

Table 15-54: SETUP ENDPOINT Command Payload

Tag	Length (bytes)	Description	Field is	Domain Version
50 _h	20	key_identifier, SHA-1 hash of the value of the BIT STRING subjectPublicKey of the target endpoint (excluding the tag, length, and number of unused bits)	mandatory	V-OD-FW
4A _h	3	2 bytes offset (unsigned big-endian) 1 byte length to be returned from confidential mailbox in AUTH1 response (persistent)	optional	N/A. Informative Only
4B _h	3	2 bytes offset (unsigned big-endian) 1 byte length to be returned from private mailbox in AUTH1 response (persistent)	optional	N/A. Informative Only
82 _h	1	00 _h (disable) / 01 _h (enable) endpoint visibility contactless interface (persistent)	optional	V-D-TX
83 _h	1	00 _h (disable) / 01 _h (enable) endpoint visibility on contactless interface (volatile)	optional	V-D-TX
84 _h	0–16	List of 1-byte transaction codes triggering a user authentication if present in AUTH0 P2 over contactless interface (persistent), see Table 9-1 , Table 9-2	optional	V-D-TX
85 _h	0–16	List of 1-byte transaction codes triggering a user authentication if present in AUTH0 P2 over wired interface (persistent), see Table 9-1 , Table 9-2	optional	V-D-TX
4E _h	0–8	key_slot to be used by receiver endpoint (1–8 byte) If tag present but empty, key_slot will be restored to initial_key_slot assigned during key creation.	optional	V-OD-FW
9B _h	1	00 _h (disable) / 01 _h (enable) endpoint visibility on wired interface (persistent)	optional	V-D-TX
9C _h	1	00 _h (disable) / 01 _h (enable) endpoint visibility on wired interface (volatile)	optional	V-D-TX

Listing 15-33: SETUP ENDPOINT Processing

```

input: key_identifier, Table 15-54
output:
begin
  if key_identifier does not match any of the existing endpoints or target endpoint is terminated
    return 6A88h
  if offset/length out of mailbox boundaries
    return 6400h
  if the selected mailbox areas sizes do not fit in AUTH1 response
    return 6400h
  if one of the transaction_code present in nvm.endpoint.transaction_code_list_cless is not present in tag 84h
    if user authentication has not been performed
      return 6982h
  if one of the transaction_code present in nvm.endpoint.transaction_code_list_wired is not present in tag 85h
    if user authentication has not been performed
      return 6982h
  atomic start
    store the persistent endpoint configuration from command payload
  if tag 4Eh is present
    if tag 4Eh is empty
      nvmwrite nvm.endpoint.key_slot ← nvm.endpoint.initial_key_slot (restore initial value from key creation)
    else
      nvmwrite nvm.endpoint.key_slot ← key_slot (assign new value provided in tag 4Eh)
  atomic commit
    apply the persistent endpoint configuration from command payload
    apply the volatile endpoint configuration from command payload
  return
end

```

15.3.2.22 SETUP INSTANCE command

This command is used to enable/disable endpoint visibility of all endpoints on the contactless interface. If visibility is disabled, the endpoints appear as non-existent on the contactless interface.

A newly created endpoint is enabled by default. Persistent configurations are applied immediately upon request (overriding current volatile configurations, if any) and are reapplied on each SE power cycle.

Volatile configurations are applied immediately upon request. When persistent and volatile configurations are present in the same command, the configurations are applied in the order mentioned in Listing 15-34 (SETUP INSTANCE processing).

command: CLA2 34 00 00 Lc [Table 15-55]

response: 90 00

The CLA2 is as defined in Table 15-3.

Table 15-55: SETUP INSTANCE Command Payload

Tag	Length (bytes)	Description	Field is	Domain Version
82 _h	1	00 _h (disable) / 01 _h (enable) all endpoints visibility on contactless interface (persistent)	optional	V-D-TX
83 _h	1	00 _h (disable) / 01 _h (enable) all endpoints visibility on contactless interface (volatile)	optional	V-D-TX

Tag	Length (bytes)	Description	Field is	Domain Version
9B _h	1	00 _h (disable) / 01 _h (enable) all endpoints visibility on wired interface (persistent)	optional	V-D-TX
9C _h	1	00 _h (disable) / 01 _h (enable) all endpoints visibility on wired interface (volatile)	optional	V-D-TX
90 _h	0	If this tag is present, all volatile configurations are cleared, and the current persistent configurations are applied.	optional	V-D-TX

Listing 15-34: SETUP INSTANCE Processing

```

1  input: Table 15-55
2  output:
3  begin
4    atomic start
5      store the persistent endpoint configuration for all endpoints in the instance
6    atomic commit
7      apply the persistent instance configuration for all endpoints in the instance
8      apply the volatile instance configuration for all endpoints in the instance
9    return
10 end

```

15.3.2.23 SIGN command

This command signs an arbitrary data field using the private key of the selected endpoint. The command can be used with or without performing a user authentication; a different “usage” value is included in the signature by the SE to differentiate these two contexts.

command: CLA2 30 [Table 15-56] 00 Lc [Table 15-57] 00

response: [Table 15-59] 90 00

Table 15-56: SIGN Command Payload

P1 value	Description	Domain Version
00 _h	User authentication is required	V-D-TX
01 _h	User authentication is not required	V-D-TX
02 _h -FF _h	RFU	N/A

The CLA2 is as defined in Table 15-3.

If P1 value is set to 0 (i.e., user authentication is required), the device will perform user authentication. The Digital Key applet obtains information on whether or not the user authentication was performed in a proprietary manner and uses the appropriate usage value for SIGN response (see Listing 15-35).

Table 15-57: SIGN Command Payload

Tag	Length (bytes)	Description	Field is	Domain Version
50 _h	20	key_identifier, SHA-1 hash of the value of the BIT STRING subjectPublicKey of the endpoint certificate that issues the attestation (excluding the tag, length, and number of unused bits)	mandatory	V-OD-FW

Tag	Length (bytes)	Description	Field is	Domain Version
58 _h	32	Arbitrary data to be signed	mandatory	N/A. Informative Only

1 *Table 15-58: SIGN Data Fields*

Tag	Length (bytes)	Description	Field is	Domain Version
41 _h	1	Version = 01 _h , version of Signature Data Fields	mandatory	V-D-TX
92 _h	8	Random	mandatory	N/A. Informative Only
5D _h	20	key_identifier, 160-bit SHA-1 hash of the value of the BIT STRING subjectPublicKey from the endpoint certificate that issued the attestation (excluding the tag, length, and number of unused bits)	mandatory	V-OD-FW
58 _h	32	arbitrary_data (provided in command)	mandatory	N/A. Informative Only
93 _h	4	usage = D074DA4F _h , if user authentication was performed usage = FC6F4C17 _h , if user authentication was not performed	mandatory	N/A. Informative Only

2
3 The following table describes the content of the SIGN response.

4 *Table 15-59: SIGN Response*

Tag	Length (bytes)	Description	Field is	Domain Version
41 _h	1	version = 01 _h , version of Signature Data Fields	mandatory	V-D-TX
92 _h	8	Random	mandatory	N/A. Informative Only
5D _h	20	key_identifier	mandatory	V-OD-FW
9E _h	64	Signature over fields from Table 15-58 with the endpoint private key	mandatory	N/A. Informative Only

5 *Listing 15-35: SIGN Processing*

```

1  input: P1, key_identifier, arbitrary_data
2  output: signature, random, key_identifier, version, (counter)
3  begin
4    if key_identifier does not match any of the existing endpoints or target endpoint is terminated
5      return 6A88h
6    if SIGN not allowed on this endpoint as per Table 15-13
7      return 6900h
8    if P1 == 00h
9      if user authentication is not performed
10     return 6985h
11     usage D074DA4Fh
12   else if P1 == 01h
13     usage FC6F4C17h
14   else
15     return 6A86h
16   version ← 01h
17   generate 8 bytes random as per Listing 15-40 to be added in data_fields according to Table 15-58
18   generate data_fields to be signed according to Table 15-58
19   version, usage, key_identifier, random, arbitrary_data

```

```

20 generate signature according to Listing 15-43 using data_fields and nvm.endpoint.SK of the current endpoint
21 return signature, random, key_identifier, version
22 end

```

15.3.2.24 *MANAGE UA command*

This command provides the user authentication status. Implementing this command is optional and the user authentication status may be provided using a proprietary method.

command: CLA2 A0 00 00 Lc [Table 15-60] 00

response: 90 00

The CLA2 is as defined in Table 15-3.

Table 15-60: *MANAGE UA Command Payload*

Tag	Length (bytes)	Description	Field is	Domain Version
80 _h	1	User authentication status 00 _h (user not authenticated) / 01 _h (user authenticated)	mandatory	V-OD-FW
81 _h	variable	Discretionary data	optional	V-OD-FW
A1 _h	variable	TLV-structured discretionary data	optional	V-OD-FW

The applet shall ignore tags 81_h and A1_h if it has no interpretation for them

15.3.2.25 *onDeselect() Event*

This event is called on explicit or implicit deselection of the applet.

Event processing pseudo-code:

Listing 15-36: *onDeselect Event Processing*

```

1 input: reason
2 output: none
3 begin
4   if contactless interface
5     (optional) send to digital key framework notify_deselect with reason
6   end

```

15.3.2.26 *CREATE RANGING KEY command*

This command generates a ranging key (pre-derived URSK) and makes it available to a UWB along with arbitrary data. This command shall only be executed after successfully receiving AUTH1 Response. If the creation of pre-derived URSK is not possible (e.g. two pre-derived URSKs already exist), the status word 6484_h is returned.

The arbitrary_data field (up to 127 bytes) is optional and reserved for future usage.

command: CLA2 71 00 00 Lc [arbitrary_data]

response: 90 00

Listing 15-37: *CREATE RANGING KEY processing.*

```

1 input: arbitrary_data
2 output: none
3 begin
4   if UWB not supported by platform
5     return 6D00h

```

```

6  if cod.transaction_state != auth1_done and cod.transaction_state != exchange_done
7      return 6400h
8  cod.transaction_state ← create_rk_done
9
10 if cod.URSK not initialized
11     interface_byte ← C3h
12     info ← cod.vehicle_ePK.x || cod.endpoint_ePK.x || cod.transaction_identifier || interface_byte ||
        cod.flag || "Volatile" || 5Ch || 02h || cod.current_protocol_version
14
15     keying_material_length ← 80
16     compute derived_keys according to Listing 15-45 using cod.Kdh, info, keying_material_length
17     URSK ← subset of derived_keys at offset 48 with length 32
18 else
19     URSK ← cod.URSK
20
21 key_identifier ← 160-bit SHA-1 hash of the value of the BIT STRING subjectPublicKey from the currently
22     selected endpoint certificate (excluding the tag, length, and number of unused bits)
23 Make URSK, cod.transaction_identifier, key_identifier, arbitrary_data available
24 if storing URSK fails because queue is full
25     return 6484h
26 if URSK is not available for other reason
27     return 6400h
28 send to digital key framework notify_URSK_created
29 return 9000h
end

```

1 15.3.2.27 DELETE RANGING KEYS command

2 This command is optionally supported to delete either a given pre-derived URSK or all pre-
3 derived URSKs associated to an endpoint. This command shall not affect the URSK used in the
4 active ranging session.

5 **command:** CLA2 41 00 00 Lc [Table 15-61]

6 **response:** 9000

7 The CLA2 is as defined in Table 15-3.

8 Table 15-61: Delete Ranging Keys Request.

Tag	Length (bytes)	Description	Field is	Domain Version
CF _h	4	UWB session ID	conditional	V-D-BT
50 _h	20	key_identifier, SHA-1 hash of the value of the BIT STRING subjectPublicKey of the target endpoint (excluding the tag, length, and number of unused bits)	conditional	V-OD-FW

9 Listing 15-38: DELETE RANGING KEYS Processing.

```

1  input: uwb_session_ID, key_identifier
2  output: none
3  begin
4      if uwb_session_ID is present
5          if uwb_session_ID does not match with any existing secure ranging session
6              return 6A88h
7          if uwb_session_ID matches the session id of a currently active secure ranging session
8              return 6400h
9          atomic start

```

```

    delete pre-derived URSK (and related data) associated with uwb_session_ID
    atomic commit
    return 9000h
if key_identifier is present
4   if key_identifier does not match with any of the existing endpoints
5       return 6A88h
6   atomic start
7       delete all pre-derived URSKs (and related data) associated with key_identifier except if URSK used
        for a currently active secure ranging session
8   atomic commit
    return 9000h
9   end
```

1 15.3.2.28 GET NOTIFICATION command

2 Implementing this command is optional and the notification(s) may be provided using a
3 proprietary method. If Option D is supported, this command shall be implemented as defined in
4 Section 15.3.1.9, in order to retrieve the notification(s) that may have been generated during the
5 processing of the previous APDU command over the wired interface and the same logical
6 channel. **command: CLA2 A1 00 00 00**

7 **response: [list of notification TLVs (7F60_h)] 90 00**

8 The CLA2 is as defined in [Table 15-3](#).

9 The command response contains a list of notifications. The coding of each notification is as
10 defined in Table 15-9. These notifications are appended to the list in the sequential order they are
11 generated. If no notification has been generated during the execution of the previous APDU
12 command, the response shall be empty.

13 15.3.2.29 CONVERT ENDPOINT command

14 This command is required to convert endpoints that cannot share keys to be capable of sharing
15 keys. This command includes an endpoint conversion request and updates endpoint parameters
16 and endpoint certificates as specified below. If the command is called on already converted
17 endpoint, SW 9000_h shall be returned.

18 **Command: CLA2 8C 00 00 Lc [Table 15-62:] 00**

19 **Response: [response_length] 90 00**

20 CLA2 is defined in Table 15-3. When the command is called, the applet first identifies the
21 endpoint using the endpoint_key_identifier from the endpoint conversion request. Then the
22 applet verifies the current value of the endpoint_conversion_counter (see section Table 15-13)
23 and the target_endpoint_conversion_counter in the command (tag 53_h).

24 If the endpoint_conversion_counter is equal to or lower than the
25 target_endpoint_conversion_counter, then the applet converts the specified endpoint to the
26 following values and sets the endpoint_conversion_counter to the
27 target_endpoint_conversion_counter value (idempotent call).

28 0001_h:

- 29 • Update option_group_1 bit4 $\leftarrow 1$, keep all other bits unchanged
- 30 • Update option_group_1 bit5 $\leftarrow 1$, keep all other bits unchanged

31 0002_h – FFFF_h:

- 32 • RFU

Note that if more than one conversion operations will be defined in the future Digital Key specification, then the framework shall call each step separately and successively to bring the applet to the final conversion state.

If the `target_endpoint_conversion_counter` in the command is set to a value not defined by this Digital Key specification, then the applet rejects execution of the command with `SW = 6200h`.

Table 15-62: Endpoint Conversion Request

Tag		Length (bytes)	Description	Field is	Domain Version
7F3A _h		variable	Endpoint conversion request	mandatory	D-VS
	50 _h	20	key_identifier, SHA-1 hash of the value of the BIT STRING subjectPublicKey of the target endpoint (excluding the tag, length, and number of unused bits)	mandatory	D-VS
	51 _h	15 or 13	not_before, DER encoded GeneralizedTime (15 bytes in length) or UTCTime (13 bytes in length) as per RFC 5280 [3]	mandatory	D-VS
	53 _h	2	Target_endpoint_conversion_counter	mandatory	D-VS

The new `endpoint_conversion_counter` can be retrieved from the internal buffer after command completion. It is provided in TLV format using tag 53_h.

The Convert Endpoint command creates a new endpoint certificate, which can be retrieved from the internal buffer on the command completion. It is provided in TLV format using Tag 7F24_h (Table 11-9). Note that the endpoint certificate has a new serial number. Also, the 'not_before' date may be different from the original endpoint certificate.

Table 15-63: CONVERT ENDPOINT Internal Buffer Content after Processing (offset 0)

Tag	Length (bytes)	Description	Field is	Domain Version
53 _h	2	New endpoint_conversion_counter	mandatory	D-VS
7F24 _h	variable	Endpoint Creation Certificate for the converted endpoint	Conditional. Present if SW 9000 _h returned	D-VS

Listing 15-39: CONVERT Endpoint Processing 1

```

1  input: conversion_request
2  output: endpoint_conversion_counter, response
3  if endpoint not found
4      return 6A82h (no response data)
5  if target_endpoint_conversion_counter (53h from endpoint conversion request) < endpoint_conversion_counter
6      clear cod.internal_buffer
7      fill cod.internal_buffer with endpoint_conversion_counter wrapped into tag 53h
8      return response_length (2 bytes big endian) | 6200h
9  if target_endpoint_conversion_counter ≥ endpoint_conversion_counter
10     if target_endpoint_conversion_counter == 0001h

```

```
11 | atomic start
12 |   endpoint_configuration.option_group_1.bit4 ← 1
13 |   endpoint_configuration.option_group_1.bit5 ← 1
14 |   endpoint_conversion_counter ← target_endpoint_conversion_counter
15 |   clear cod.internal_buffer
16 |   fill cod.internal_buffer with endpoint_conversion_counter wrapped into tag 53h
17 |   re-generate endpoint certification, wrap into tag 7F24h and store in cod.internal_buffer
18 |   response_length = length written in cod.internal_buffer
19 | atomic commit
20 |   return response_length (2 bytes big endian) | 9000h
21 | if target_endpoint_conversion_counter > 0001h // reserved for future
22 |   clear cod.internal_buffer
23 |   fill cod.internal_buffer with endpoint_conversion_counter wrapped into tag 53h
24 |   return response_length (2 bytes big endian) | 6200h
25 | end
```

1 15.3.3 Security

2 15.3.3.1 Cryptography Algorithms

3 Listing 15-40: Generate Random

```
1 | input: length
2 | output: random
3 | begin
4 |   select random generator compliant with 'AIS31' standard
5 |   random = buffer filled with random bytes for the size indicated in input parameters
6 |   return random
7 | end
```

4 Listing 15-41: Generate Key Pair

```
1 | input: none
2 | output: ecc.PK.x, ecc.PK.y, ecc.PK, ecc.SK
3 | begin
4 |   set curve_parameters ← 'ECC NIST P-256' as per [8]
5 |   generate ecc.PK.x, ecc.PK.y, ecc.SK
6 |   ecc.PK = (ecc.PK.x, ecc.PK.y)
7 |   return ecc.PK.x (32 bytes), ecc.PK.y (32 bytes), ecc.SK (32 bytes)
8 | end
```

5 Listing 15-42: Generate Identifier

```
1 | input: buffer
2 | output: identifier
3 | begin
4 |   hash ← SHA-1(buffer)
5 |   identifier = 6 first bytes of hash
6 |   return identifier
7 | end
```

6 Listing 15-43: Generate Attestation Signature

```
1 | input: data_fields, private_key
2 | output: signature
3 | begin
4 |   set curve_parameters ← 'ECC NIST P-256' as per [8]
5 |   signature ← ECDSA(curve_parameters, private_key, data_fields) using SHA-256 as per [8]
6 |   return signature (64 bytes)
7 | end
```


1 *Listing 15-44: Verify Attestation Signature*

```
1 input: data_fields, public_key, signature
2 output: boolean
3 begin
4   hash ← SHA-256(data_fields)
5   set curve parameters ← 'ECC NIST P-256' as per [8]
6   boolean ← Verify(public_key, hash, signature)
7   return boolean (true/false)
8 end
```

2 *Listing 15-45: Compute Shared Key with Diffie-Hellman*

```
1 input: ePK, eSK, (transaction_identifier)
2 output: Kdh
3 begin
4   Compute the steps indicated by Section 4.3 of BSI TR-03111 [6] with the following mapping:
5   KeyAgreementProtocol : ECKA-DH
6   (p, a, b, G, n, h) : ECC NIST P-256 Curve parameters as per [8]
7   d̂ : eSK
8   P̂ : ePK
9   SAB : key agreement output (shared secret point)
10  ZAB : computed from SAB as per Section 3.1.3 of BSI TR-03111 [6]
11  key derivation input : ZAB
12  Key Derivation Function: KDFX9.63 as per Section 4.3.3 of BSI TR-03111 [6]
13  K : 256
14  H : SHA-256
15  SharedInfo : (transaction_identifier)
16  KeyData : Kdh (32 bytes)
17 end
```

3 *Listing 15-46: Key Derivation*

```
1 input: input_keying_material, info, keying_material_length
2 output: derived_keys
3 begin
4   Compute the steps indicated by Section 2 of RFC 5869 [13] with the following mapping:
5   Hash : SHA-256
6   IKM : input_keying_material
7   salt : NULL
8   info : info
9   L : keying_material_length
10  OKM : derived_keys
11 end
```

4 *Listing 15-47: Secure Channel Command Decryption and Authentication*

```
1 input: encrypted_command_payload, Kmac, Kenc, counter, mac_chaining_in
2 output: decrypted_command_payload, mac_chaining_out
3 begin
4   Compute the steps indicated by Section 6.2.6 of GPC_SPE_014 [7] with the following mapping:
5   CommandDataField - Plain Text : payload
6   PaddedCounterBlock : 00000000000000000000000000000000h || counter (1 byte, up to max FFh)
7   S-ENC : Kenc
8   CipheredCommandDataField : encrypted_command_payload
9   CommandDataField - PlainText : decrypted_command_payload
10
11  Compute the steps indicated by Section 6.2.4 of GPC_SPE_014 [7] with the following mapping:
12  Variationfromspec : MAC plaintext payload only includes command data field (CLA, INS, P1, P2, Lc, Le are omitted)
13  CommandDataField : encrypted_command_payload
14  MACChainingValue : mac_chaining_in (16 bytes)
```

```

15  NewMACChainingValue : mac_chaining_out (16 bytes)
16  S-MAC : Kmac
17  end

```

1 *Listing 15-48: Secure Channel Response Encryption and Authentication*

```

1  input: payload, Krmac, Kenc, counter, mac_chaining_in
2  output: encrypted_payload, mac
3  begin
4  Compute the steps indicated by Section 6.2.7 of GPC_SPE_014 [7] with the following mapping:
5  ResponseDataField – PlainText : payload
6  PaddedCounterBlock : 80000000000000000000000000000000h || counter (1 byte, up to max FFh)
7  S-ENC : Kenc
8  CipheredResponseDataField : encrypted_payload
9
10 Compute the steps indicated by Section 6.2.5 of GPC_SPE_014 [7] with the following mapping:
11 Variationfromspec : MAC plaintext payload only includes MAC Chaining Value || encrypted payload, (SW is omitted)
12 ResponseDataField : encrypted_payload
13 MACChainingValue : mac_chaining_in (16 bytes)
14 S-RMAC : Krmac
15 R-MAC : mac (8 bytes)
16 end

```

2 *Listing 15-49: Derive KEenc, KEmac*

```

1  input: KEseed_half, sender_ePK, receiver_ePK, derivation, receiver_PK
2  output: KEenc, KEmac
3  begin
4  Compute the steps indicated by Section 4.1 of NIST SP800-108r1 [4] with the following mapping:
5  PRF : CMAC as defined by NIST SP800-38B using AES-128 block cipher [5]
6  h : 128
7  r : 8
8  KIN : KEseed_half
9  Label : 00000000000000000000000000000000h concatenated with 08h /12h for KEenc/KEmac generation
10 Context : receiver_ePK.x || sender_ePK.x || receiver_PK.x
11 L : 0080h (AES-128)
12 PRF (KIN, Label || 00h || L || i || Context) : concatenation order
13 KOUT : KEenc(16 bytes) or KEmac(16 bytes)
14 end

```

3 *Listing 15-50: Confidential Mailbox Encryption and Authentication*

```

1  input: payload, KEenc, KEmac
2  output: encrypted_payload, mac
3  begin
4  Compute the steps indicated by Section 6.2.6 of GPC_SPE_014 [7] with the following mapping:
5  CommandDataField - PlainText : payload
6  PaddedCounterBlock : 00000000000000000000000000000001h
7  S-ENC : KEenc
8  CipheredCommandDataField : encrypted_payload
9
10 Compute the steps indicated by Section 6.2.4 of GPC_SPE_014 [7] with the following mapping:
11 Exception: The header [84h Ins P1 P2 Lc] shall not be included in the CMAC input,
12 only MAC Chaining Value and payload are used as input to the CMAC block.
13 CommandDataField : encrypted_payload
14 MACChainingValue : 00000000000000000000000000000000h
15 S-MAC : KEmac
16 C-MAC : mac (8 bytes)
17 end

```

Listing 15-51: Compute DeviceCryptogram

```
input: KCmac, endpoint_PK.x, vehicle_PK.x, transaction_identifier, vehicle_identifier
output: cryptogram
begin
  Compute the steps indicated by Section 4.1 of NIST SP800-108r1 [4] with the following mapping:
  PRF : CMAC as defined by NIST SP800-38B using AES-128 block cipher [5]
  h : 128
  r : 8
  KIN : KCmac
  Label : 000000000000000000000000 concatenated with 32n
  Context : vehicle_PK.x || endpoint_PK.x || transaction_identifier || vehicle_identifier
  L : 0080n (AES-128)
  PRF(KIN, Label || 00\hex || L || i || Context) : concatenation order
  KOUT : cryptogram(16 bytes)
end
```

15.3.4 Optimization

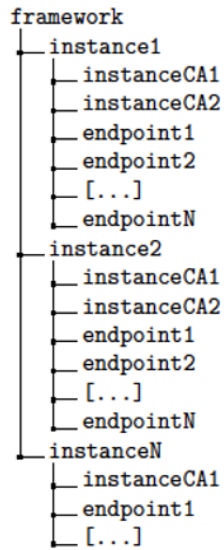
This section provides a non-exhaustive list of optimizations that may improve the applet performance. These optimizations depend on platform capabilities. The section is provided for informational purposes only.

1. The ephemeral public key generated during the AUTH0 command could be pre-generated at the end of each transaction (onDeselect). In order to avoid transaction delays in case of communication errors, several pre-generated ephemeral keys could be used until fallback to runtime generation. For transactions over wired interface, ephemeral key pairs should be generated at runtime instead of from the pool of pre-generated keys because of more relaxed timing constraints.
2. An ephemeral public key generated during the AUTH0 command could be re-used for a limited amount of time (e.g., a few seconds) in order to avoid transaction-time key pair generation on early failed transactions. However, an ephemeral key pair should be deleted after being used in an ECDH operation.
3. The generation of the endpoint signature could be done in parallel with reader processing in the vehicle after sending the AUTH0 response.
4. The next Kpersistent could be generated in parallel with reader processing in the vehicle after sending the AUTH0 response.
5. The Kmac, Kenc, and Krmac could be generated in parallel with reader processing in the vehicle after sending the AUTH0 response.
6. The AUTH1 response wrapped in the secure channel could be prepared in parallel with reader processing in the vehicle after sending the AUTH0 response.

15.4 Helper Method

This section describes groups of APDU commands to be referred as a single functional block in other specifications. The section is provided for informational purposes only.

1 15.4.1 Hierarchy



2

3 15.4.1.1 Types

4 For the purpose of description, variable names are prefixed with the following types:

- 5 • **bool_**: A boolean
- 6 • **bytes_**: An array of 8-bit bytes
- 7 • **uX_**: An unsigned value of X bits
- 8 • **obj**: An object
- 9 • **type[]**: A list of objects of the defined type

10 15.4.1.2 *framework.createInstance*

11 Create a new instance of the Digital Key applet. The instance can contain several endpoints.

12 **Inputs:**

13 **bytes_AID**: The instance AID.

14 **u32_endpointCountMax**: The maximum number of endpoints.

15 **u32_instanceCACountMax**: The maximum number of Instance CAs.

16 **u32_internalBufferSize**: The size of internal buffer.

17 **u32_maxAllocatablePrivateMailboxSize**: The maximum private mailbox size of an endpoint.

18 **u32_maxAllocatableConfidentialMailboxSize**: The maximum confidential mailbox size of an endpoint.

20 **Outputs:**

21 **obj_instance**: An object representing the Digital Key applet instance created

22 *Listing 15-52: framework.createInstance Processing*

1	input: bytes_AID, u32_endpointCountMax, u32_instanceCACountMax, u32_internalBufferSize
2	output: obj_instance
3	begin

```
4 request instance provisioning to the Device OEM Server with input parameters
5 create a handle representing the Digital Key applet instance, obj_instance.
6 return obj_instance
7 end
```

1 15.4.1.3 *framework.getInstance*

2 Retrieve an object representing the Digital Key applet instance.

3 **Inputs:**

4 **bytes_AID:** The instance AID

5 **Outputs:**

6 **obj_instance:** An object representing the selected instance

7 *Listing 15-53: framework.getInstance Processing*

```
1 input: bytes_AID
2 output: obj_instance
3 begin
4 create a handle representing the Digital Key applet instance, obj_instance
5 return obj_instance
6 end
```

8 15.4.1.4 *framework.view*

9 Get AID list of installed instance.

10 **Inputs:**

11 n/a

12 **Outputs:**

13 **bytes_instanceAID[]:** AID list of the instances installed

14 *Listing 15-54: framework.view Processing*

```
1 input: n/a
2 output: bytes_instanceAID[]
3 begin
4 return installed instance AID list
5 end
```

15 15.4.1.5 *framework.deleteInstance*

16 Delete a Digital Key applet instance and all related resources. The method described is currently
17 not called in this specification. It is included for completeness and potential usage by other
18 specifications.

19 **Inputs:**

20 **obj.instance:** Object representing the instance

21 *Listing 15-55: framework.deleteInstance Processing*

```
1 input: obj.instance
2 output: n/a
3 begin
```

```
4 request instance deletion to the Device OEM Server with input parameters
5 end
```

1 15.4.1.6 *instance.view*

2 Retrieve the instance related information. This method is called over an instance object
3 (obj_instance).

4 **Outputs:**

- 5 **u8_endpointCount:** The number of endpoints created.
- 6 **u8_endpointCountMax:** The maximum number of endpoints
- 7 **u16_internalBufferSize:** The size of the internal buffer
- 8 **bytes_keyIdentifier[]:** A list of key identifiers present on the instance
- 9 **bytes_instanceCAIdentifier[]:** A list of Instance CA identifiers present on the instance

10 *Listing 15-56: instance.view Processing*

```
1 input: obj_instance
2 output: u8_endpointCount,
3         u8_endpointCountMax,
4         u16_internalBufferSize,
5         bytes_keyIdentifier[],
6         bytes_instanceCAIdentifier[]
7 begin
8 retrieve instance informations using command in Section 15.3.2.8
9 return u8_endpointCount,
10        u8_endpointCountMax,
11        u16_internalBufferSize,
12        bytes_keyIdentifier[],
13        bytes_instanceCAIdentifier[]
14 end
```

11 15.4.1.7 *instance.createEndpoint*

12 Create an endpoint on the selected instance. This method is called over an instance object
13 (obj_instance).

14 If bool_onlineCertificate is set, the endpoint creation certificate is signed by the Device OEM
15 CA instead of the Instance CA. The device needs to be able to reach the Device OEM Server for
16 this feature to be available.

17 **Inputs:**

- 18 **bytes_vehicleIdentifier:** The vehicle identifier
- 19 **bytes_endpointIdentifier:** The endpoint_identifier
- 20 **bytes_instanceCAIdentifier:** The identifier of the Instance CA to be used for key
21 attestation
- 22 **u8_optionGroup1:** The option group 1
- 23 **u8_optionGroup2:** The option group 2
- 24 **u16_protocolVersion:** The Digital Key applet protocol version
- 25 **bytes_vehiclePK:** The vehicle's public key

bytes_notBefore: DER encoded GeneralizedTime.
bytes_notAfter: DER encoded GeneralizedTime.
bytes_authorizedPK[]: List of public keys of the authorized CAs ($n \times 65$ bytes, $n < 6$).
u16_confidentialMailboxSize: The confidential_mailbox_size (optional)
u16_privateMailboxSize: The private_mailbox_size (optional)
bytes_keySlot: The key slot (optional)
u32_counterLimit: The signature counter limit (deprecated)

Outputs:

obj_endpoint: An object representing the created endpoint

Listing 15-57: instance.createEndpoint Processing

```
1 input: input list
2 output: obj_endpoint
3 begin
4   generate input_data as per Table 15-7
5   if input_data > 255
6     send the WRITE BUFFER command as described in Section 15.3.2.14 with input input_data
7     send the CREATE ENDPOINT command as described Section 15.3.2.4 without payload
8   else
9     send the CREATE ENDPOINT command as described Section 15.3.2.4 with input_data as payload
10    create a handle representing the created endpoint, obj_endpoint
11  return obj_endpoint
12 end
```

15.4.1.8 instance.deleteEndpoint

Delete an endpoint on a specific instance. This method is called over an instance object (obj_instance).

This method is currently not called in this specification. It is included for completeness and potential usage by other specifications.

Inputs:

bytes_keyIdentifier: The key identifier

Listing 15-58: instance.deleteEndpoint Processing

```
1 input: bytes_keyIdentifier
2 output: n/a
3 begin
4   send the DELETE ENDPOINT command as described in Section 15.3.2.6 with input_data as payload
5   delete endpoint corresponding data on digital key framework
6 end
```

15.4.1.9 instance.getEndpoint

Obtain a handle on an endpoint. This method is called over an instance object (obj_instance).

Inputs:

bytes_keyIdentifier: The key identifier

Outputs:

obj_endpoint: An object representing the endpoint

Listing 15-59: instance.getEndpoint Processing

```
1 input: bytes_keyIdentifier
2 output: obj_endpoint
3 begin
4   create a handle representing the endpoint, obj_endpoint.
5   return obj_endpoint
6 end
```

15.4.1.10 *instance.setParameters*

Set instance parameters. This method is called over an instance object (obj_instance).

This method is currently not called in this specification. It is included for completeness and potential usage by other specifications.

Inputs:

bool_cless_visibility_persistent: If set to false, all endpoints are invisible from the contactless interface. This configuration is persistent across SE reboots.

bool_cless_visibility_volatile: If set to false, all endpoints are invisible from the contactless interface. This configuration is valid for the current power on session.

bool_wired_visibility_persistent: If set to false, all endpoints are invisible from the wired interface on AUTH0 command. This configuration is persistent across SE reboots.

bool_wired_visibility_volatile: If set to false, all endpoints are invisible from the wired interface on AUTH0 command. This configuration is valid for the current power on session.

Listing 15-60: instance.setParameters Processing

```
1 input: obj_instance, parameters
2 output: n/a
3 begin
4   set instance parameters using command as described in Section 15.3.2.22
5 end
```

15.4.1.11 *endpoint.setParameters*

Set endpoint parameters. This method is called over an endpoint object (obj_endpoint). Note that all inputs listed below in this subsection are optional.

Inputs:

u16_offset_confidential: The offset from which data contained in confidential mailbox is returned in AUTH1.

u8_length_confidential: The length of confidential data returned in AUTH1.

u16_offset_private: The offset from which data contained in private mailbox are returned in AUTH1.

u8_length_private The length of private data returned in AUTH1.

bool_cless_visibility_persistent: If set to false, the endpoint is not visible from the contactless interface. This configuration is persistent across SE reboots.

bool_cless_visibility_volatil: If set to false, the endpoint is not visible from the contactless interface. This configuration is valid for the current power-on session.

bytes_cless_transaction_codes: List of 1-byte transaction codes triggering a user authentication if present in AUTH0 P2 over contactless interface; see [Table 9-1](#) and [Table 9-2](#).

bytes_wired_transaction_codes: List of 1-byte transaction codes triggering a user authentication if present in AUTH0 P2 over contactless interface; see [Table 9-1](#) and [Table 9-2](#).

bool_wired_visibility_persistent: If set to false, the endpoint is not visible from the wired interface on AUTH0 command. This configuration is persistent across SE reboots.

bool_wired_visibility_volatile: If set to false, the endpoint is not visible from the wired interface on AUTH0 command. This configuration is valid for the current power-on session.

Listing 15-61: endpoint.setParameters Processing

```
1  input: obj_endpoint, parameters
2  output: n/a
3  begin
4    if bytes_cless_transaction_codes is missing any transaction code present in current configuration
5      perform user authentication
6    if bytes_wired_transaction_codes is missing any transaction code present in current configuration
7      perform user authentication
8    set endpoint parameters using command in Section 15.3.2.21
9  end
```

15.4.1.12 endpoint.getCertificate

Retrieve an endpoint certificate. This method is called over an endpoint object (obj_endpoint).

Input:

bool_onlineCertificate request: Attestation created by Device OEM CA (true) or by Instance CA (false).

Output:

bytes_endpointAttestation: An endpoint certificate as per Listing 15-5.

Listing 15-62: endpoint.getCertificate Processing

```
1  input: bool_onlineCertificate
2  output: bytes_endpointCertificate
3  begin
4    if bool_onlineCertificate = true
5      if endpoint certificate issued by device OEM exists
6        return endpoint certificate issued by device OEM stored in digital key framework
7    else
8      send VIEW command to retrieve endpoint certificate
9    return endpoint certificate issued by instance CA
10 end
```

1 15.4.1.13 *endpoint.terminate*

2 Terminate an endpoint and obtain a termination attestation. This method is called over an
3 endpoint object (obj_endpoint).

4 **Inputs:**

5 **bytes_nonce:** Termination nonce (16 bytes)

6 **bytes_arbitrary_data:** Arbitrary data (optional, 1–40 bytes).

7 **Output:**

8 **obj_attestation:** A key termination attestation

9 *Listing 15-63: endpoint.terminate Processing*

```
1 input: bytes_nonce, (bytes_arbitrary_data)
2 output: bytes_terminationAttestation
3 begin
4   send TERMINATE ENDPOINT command to retrieve attestation as described
5   In Table 15-20 using bytes_nonce (and bytes_arbitrary_data)
6   return bytes_terminationAttestation
7 end
```

10 15.4.1.14 *endpoint.authorize*

11 Sign an externally received endpoint attestation along with arbitrary data and optionally export
12 confidential mailbox data. This method is called over an endpoint object (obj_endpoint).

13 *Table 15-64: Receiver Endpoint Key Attestation Signed by Sender*

Tag	Length (bytes)	Description	Field is
7F25 _h	variable	Receiving Endpoint Key Attestation	mandatory
content of Table 15-24			mandatory
9E _h	64	Signature with sender endpoint private key over Table 15-24	mandatory

14 **Inputs:**

15 **u16_offset:** Offset of confidential mailbox to be retrieved (optional)

16 **u16_length:** Length of confidential mailbox to be retrieved (optional)

17 **bytes_arbitraryData:** Arbitrary data to be signed (optional)

18 **bytes_externalCertificate:** External CA certificate of the receiver as per [Listing 15-13](#)
19 (optional)

20 **bytes_instanceCertificate Instance:** CA Certificate of the receiver as per [Listing 15-16](#)
21 (optional)

22 **bytes_encryptionKeyAttestation:** Encryption key attestation of the receiver as per [Table](#)
23 [15-49](#) (optional)

24 **bytes_endpointCreationCertificate:** Endpoint creation certificate of the receiver as per
25 Listing 15-5

Outputs:

bytes_attestation: Attestation signed by the sender endpoint containing arbitrary data and public key of the receiving endpoint

bytes_encryptedData: Data encrypted for the receiver confidential mailbox

bytes_encSenderePK: Encryption public key of the sender

Listing 15-64: endpoint.authorize Processing

```
1 input: (u16_offset, u16_length, bytes_arbitraryData, bytes_instanceCertificate,  
2       bytes_encryptionKeyAttestation, bytes_externalCertificate,) bytes_endpointCreationCertificate  
3 output: bytes_attestation, bytes_encryptedData, bytes_encSenderePK  
4 begin  
5   perform user authentication  
6   send WRITE BUFFER command in order to send bytes_instanceCertificate,  
7       bytes_encryptionKeyAttestation, bytes_endpointCreationCertificate  
8   send AUTHORIZE ENDPOINT command with u16_offset, u16_length, bytes_arbitraryData  
9   bytes_attestation = prepare table as per Table 15-64  
10  bytes_encryptedData = content of tag 4Ah from Table 15-26  
11  bytes_encSenderePK = content of tag 97h from Table 15-26  
13 return bytes_attestation, bytes_encryptedData, bytes_encSenderePK  
14 end
```

15.4.1.15 *endpoint.createEncryptionKey*

Retrieve a single usage encryption key for confidential mailbox writing by other entities. This method is called over an endpoint object (obj_endpoint).

Output:

bytes_attestation: Attestation containing encryption key as per Table 15-49

Listing 15-65: endpoint.createEncryptionKey Processing

```
1 input: none  
2 output: bytes_attestation  
3 begin  
4   send CREATE ENCRYPTION KEY command to retrieve attestation as described in Section 15.3.2.17  
5   return attestation  
6 end
```

15.4.1.16 *endpoint.setConfidentialData*

Write data in the confidential mailbox. This method is called over an endpoint object (obj_endpoint).

Inputs:

bytes_encsenderpk: Public ephemeral key of the sender

bytes_encdata: Encrypted data to be written in private mailbox

u16_offset: Offset of data to be written

Listing 15-66: endpoint.setConfidentialData Processing

```
1 input: bytes_encsenderpk, bytes_encdata, u16_offset, u16_offset  
2 output: n/a  
3 begin
```

```
4  send WRITE BUFFER command to write bytes_encsenderpk and bytes_encdata in internal buffer as described in Section
   15.3.2.14
5  send SET CONFIDENTIAL DATA command to write in confidential mailbox as described in Section 15.3.2.20
6  end
```

1 15.4.1.17 *endpoint.getPrivateData*

2 Retrieve data from the private mailbox. This method is called over an endpoint object
3 (obj_endpoint).

4 **Inputs:**

5 **u16_offset:** Offset of private mailbox to be retrieved

6 **u16_length:** Length of private mailbox to be retrieved

7 *Listing 15-67: endpoint.getPrivateData Processing*

```
1  input: u16_offset
2  output: u16_length
3  begin
4  execute one or multiple GET PRIVATE DATA command as described in Section 15.3.2.18 to retrieve the
   requested data
5  end
```

8 15.4.1.18 *endpoint.setPrivateData*

9 Write data in the private mailbox. This method is called over an endpoint object (obj_endpoint).

10 **Inputs:**

11 **u16_offset:** Offset to start writing in private mailbox

12 **bytes_data:** Data to be written in private mailbox

13 *Listing 15-68: endpoint.setPrivateData Processing*

```
1  input: u16_offset, bytes_data
2  output: n/a
3  begin
4  Send SET PRIVATE DATA command to write in private mailbox as described in Section 15.3.2.19
5  end
```

14 15.4.1.19 *endpoint.sign*

15 Obtain a signature on arbitrary data. This method is called over an endpoint object
16 (obj_endpoint).

17 *Table 15-65: Arbitrary Data Attestation*

Tag	Length (bytes)	Description	Field is
7F2D _h	variable	Arbitrary Data Attestation	mandatory
Content of Table 15-58			mandatory
9E _h	64	signature with endpoint.SK over fields from Table 15-58	mandatory

2 **bytes_arbitraryData:** Arbitrary data to be signed

4 **bytes_attestation:** An attestation as described in Table 15-65

```

1  input: bytes_arbitraryData
2  output: bytes_attestation
3  begin
4      perform user authentication if required
5      send SIGN command to retrieve attestation as described in Table 15-65
6      return attestation
7  end

```

```

7   Start the mutual authentication procedure. This method is called over an endpoint object
8   (obj_endpoint).

```

```
12 bytes_vehicleIdentifier: Vehicle identifier
```

15 **u16_protocolversion:** Digital Key applet protocol version supported by endpoint

```

1 input: bytes_epkinput, bytes_transactionIdentifier, bytes_vehicleIdentifier
2 output: bytes_epkoutput, u16_protocolversion
3 begin
4     send SELECT command using the AID of the obj_endpoint to retrieve u16_protocolversion
5     send AUTH0 command as described in Section 15.3.2.9 using bytes_epkinput, bytes_transactionIdentifier,
    bytes_vehicleIdentifier
6     P1 parameters are Standard Transaction requested, User Authentication not requested and
7     retrieve bytes_epkoutput
8     return bytes_epkoutput, u16_protocolversion
9 end

```

18 The second step of the mutual authentication procedure, `endpoint.auth0` shall have been executed
19 previously. This method is called over an endpoint object (`obj_endpoint`).

21 **bytes_signature**: Signature of the third party entity wishing to authenticate

Output:

bytes_encryptedPayload: Encrypted payload

Listing 15-71: endpoint.auth1 Processing

```
1 input: bytes_signature
2 output: bytes_encryptedPayload
3 begin
4   send AUTH1 command using bytes_signature as described in Section 15.3.2.10
5   return bytes_encryptedPayload
6 end
```

15.4.1.22 endpoint.exchange

Read and write data in mailboxes upon establishment of a secure channel, provided that endpoint.auth0 (and endpoint.auth1 depending on endpoint configuration) have been successfully executed. This method is called over an endpoint object (obj_endpoint).

Input:

bytes_encryptedInput: Encrypted payload

Output:

bytes_encryptedOutput: Encrypted payload

Listing 15-72: endpoint.exchange Processing

```
1 input: bytes_encryptedInput
2 output: bytes_encryptedOutput
3 begin
4   send EXCHANGE command using bytes_encryptedInput as described in Section 15.3.2.15
5   return bytes_encryptedOutput
6 end
```

15.4.1.23 endpoint.createRangingKey

Generate a ranging key and make it available to UWB. Shall only be called after successfully receiving AUTH1 Response.

Inputs:

bytes_arbitraryData: arbitrary data.

Listing 15-73: endpoint.createRangingKey processing.

```
1 input: bytes_arbitraryData
2 output: none
3 begin
4   send CREATE RANGING KEY command using bytes_arbitraryData as described in Section 15.3.2.26
5 end
```

15.5 Vehicle Implementation

15.5.1 Security Guidelines

The following items describe important implementation guidelines for the vehicle.

1. During the standard transaction, the vehicle shall always verify the endpoint signature (endpoint_sig) before any other operation. A successful verification of this signature is the only way for the vehicle to authenticate the endpoint.
2. During the standard transaction, the vehicle shall never modify its persistent memory before having successfully verified the endpoint signature (endpoint_sig).
3. During the standard transaction, the vehicle shall never write data in the endpoint confidential or private mailbox before having successfully verified the endpoint signature (endpoint_sig).
4. During the standard and fast transactions, the vehicle shall verify that the use of contactless interface is indicated in the response to the AUTH0 and AUTH1 command when the transaction is performed over the NFC interface.[WCC1]

15.5.2 Optimizations

The following items describe recommended implementation guidelines for the vehicle.

1. One or several vehicle ephemeral keys can be pre-generated such that a fresh ephemeral key is immediately ready when the next transaction starts.
2. If a vehicle only supports the fast transaction, a random number can be generated instead of an ephemeral key pair.

16 CERTIFICATE

16.1 General

The PKI model described in this section is based on PKI mechanisms to build a chain of trust down to a single Digital Key. The following certificate chains are defined:

1. Device certificate chain
2. Vehicle certificate chain
3. Owner pairing key attestation chain
4. Key sharing key attestation chain

The objective is to provide full offline capability through the verification chain.

Certificate expiry shall be checked by the KTS. If technical capabilities allow, the Digital Key applet, vehicle, and owner device should also verify certificate expiry.

In this specification, two variants of the SE root of trust certificate chain model are provided.

- **Variant 1:** SE root of trust based on CASD: In this model (see Figure 16-1), the SE root of trust is managed by the SE Root CA implemented by the CASD.
- **Variant 2:** SE root of trust based on DK applet associated security domain: In this model (see Figure 16-2), the SE root of trust is managed using the secure channel of the security domain associated with the DK applet.

The provisioning of the root of trust in the SE is out of scope of this specification.

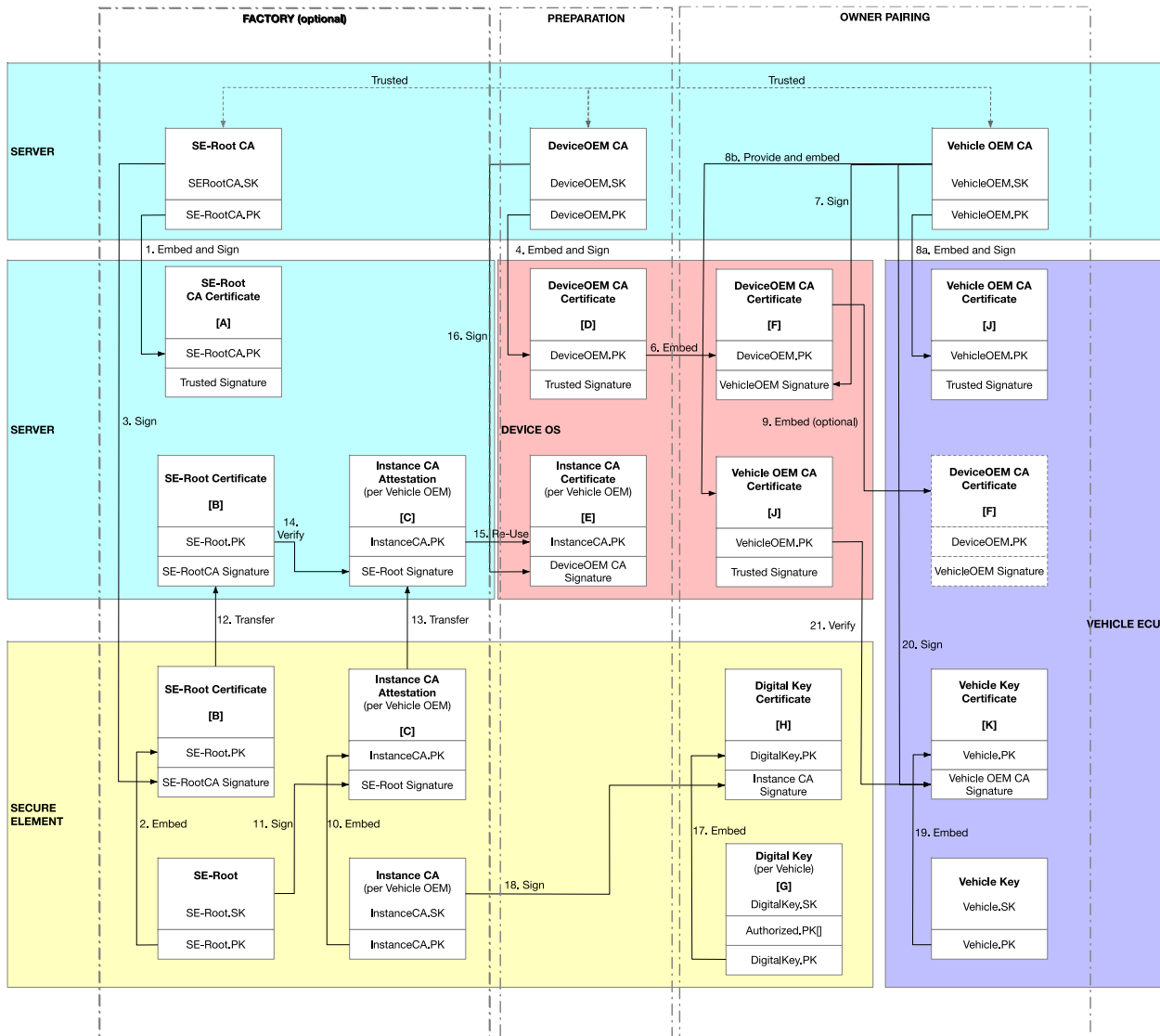
16.2 Certificates and Relationships

Figure 16-1 provides an overview of the overall certification chains when Variant 1 is implemented.

The Device OEM may choose to directly embed the Vehicle OEM CA Certificate [J] in the device OS as depicted in Figure 16-1. Alternatively, the Device OEM may choose to embed the Vehicle OEM CA Certificate signed by the Device OEM CA [M] in the device OS.

1

Figure 16-1: Variant 1 Certification Chain Model

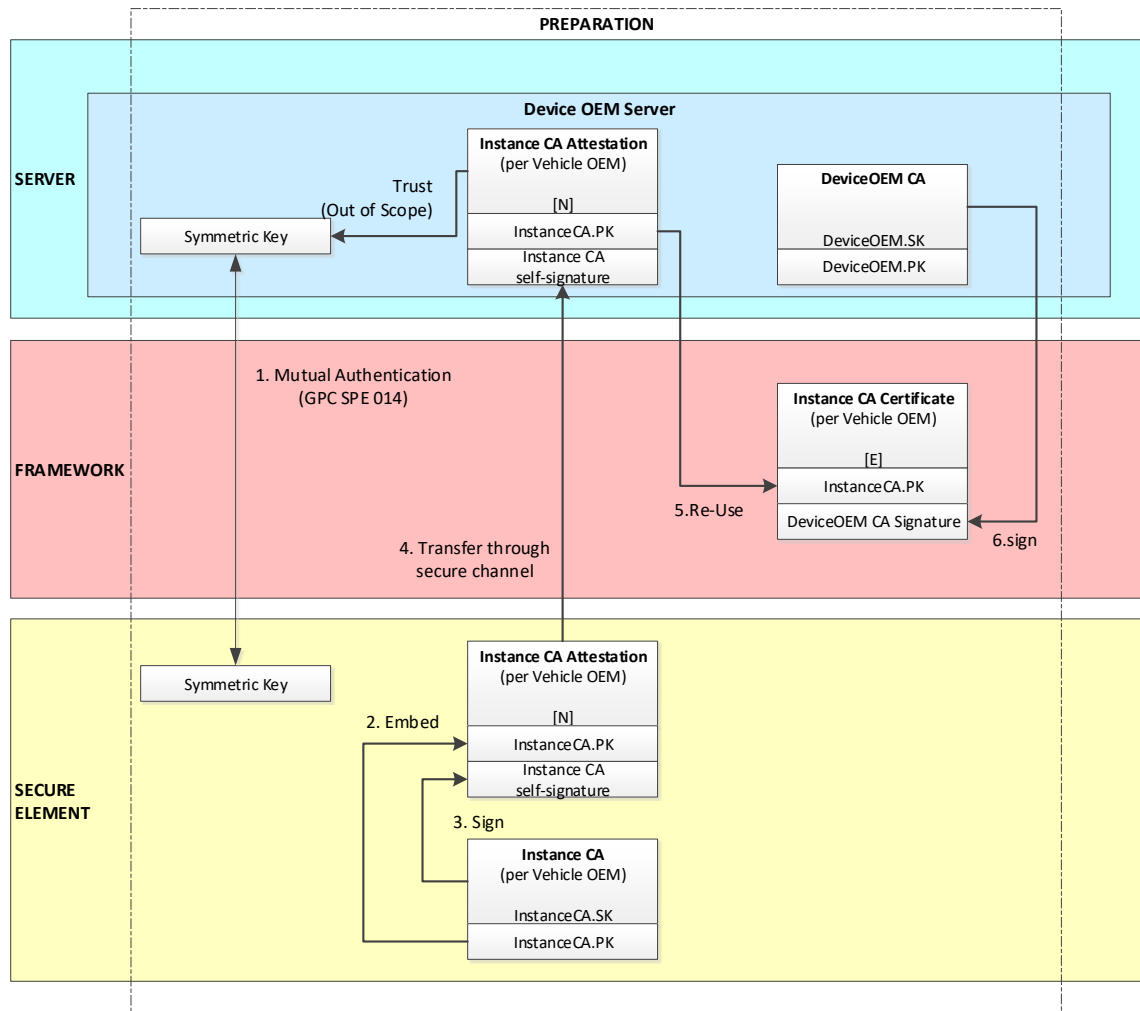


2

3 Figure 16-2 provides an adaptation when Variant 2 is implemented.

1

Figure 16-2: Variant 2 Certification Chain Model



2

3 [A] - SE Root CA Certificate – Variant 1

4 [B] - SE Root Certificate – Variant 1

5 [C] - Instance CA Attestation (signed by SE Root) per Vehicle OEM – Variant 1

6 [D] - Device OEM CA Certificate

7 [E] - Instance CA Certificate (signed by Device OEM CA) per Vehicle OEM

8 [F] - Device OEM CA Certificate (signed by Vehicle OEM CA)

9 [G] - Digital Key per Vehicle

10 [H] - Digital Key Certificate

11 [J] - Vehicle OEM CA Certificate

12 [K] - Vehicle Public Key Certificate

13 [L] - Digital Key Creation Data

14 [M] - Vehicle OEM CA Certificate (signed by Device OEM CA)

15 [N] - Instance CA Attestation (self-signed) per Vehicle OEM – Variant 2

16.2.1 [A] – SE Root CA Certificate

This certificate applies to Variant 1.
This certificate is provided by the entity that controls the root key pair in the SE, e.g., the SE manufacturer. It is trusted by the Device OEM CA.
The SE Root CA private key has signed the SE Root Certificates [B] embedded in the SE. Those certificates include immutable identifiers that could allow device tracking.
It is mandatory for every Digital Key-eligible Device OEM to be able to validate SE root signatures from the SEs on its devices.

16.2.2 [B] – SE Root Certificate

This certificate applies to Variant 1.
The SE Root Certificate [B] is generated securely and attests to the identity of the SE. [B] is signed by the SE Root CA. [B] is used to verify the Instance CA Attestation [C] created by the Digital Key applet instance. If [C] is successfully verified by [B], the Device OEM signed Instance CA Certificate [E] is issued by the Device OEM Server and stored in the device OS of the corresponding device.
Within the SE, [B] is stored in the CASD. The provisioning of the [B] in the SE is out of scope of this specification.

16.2.3 [C] – Instance CA Attestation (signed by SE Root) per Vehicle OEM

This certificate applies to Variant 1.
The Instance CA Attestation [C] is generated after an Instance CA is created by the Digital Key applet instance. [C] includes the public key of the Instance CA and is signed by the private key of the SE root. One Instance CA is created per Vehicle OEM. The Device OEM CA verifies and signs [C] with its private key to issue a new Instance CA Certificate [E] (See Section 16.2.5). [E] is verifiable using the Device OEM CA Certificate [D] and Vehicle OEM signed Device OEM CA Certificate [F].
Within the SE, the private key of the SE root is stored in the CASD. The provisioning of this private key is out of scope of this specification.
The signature of [C] by the private key of the SE root may rely on a proprietary API implemented by the CASD or may rely on the CASD signature service available through AuthoritySignature interface as defined in [15].

16.2.4 [D] – Device OEM CA Certificate

This certificate is the Device OEM root of trust for the Digital Key system. This certificate is trusted by the Device OEM and all Vehicle OEMs. The Vehicle OEM receives a CSR from the Device OEM in order to countersign the Device OEM CA Certificate [D]. This then becomes a Vehicle OEM signed Device OEM CA Certificate [F] with the same Device OEM public key.
Depending on the Device OEM's implementation or the Digital Key deployment model, the Device OEM CA Certificate [D] may be stored in the device OS or in the SE.

16.2.5 [E] – Instance CA Certificate (Signed by Device OEM) per Vehicle OEM

The Device OEM-signed Instance CA Certificate [E] contains the public key of the Instance CA Attestation [C] (Variant 1) or [N] (Variant 2) and is signed by the private key of the Device OEM CA. [E] allows the verification of the Instance CA Attestation using a Device OEM CA Certificate [F]. Using the Device OEM CA signature removes the requirements of otherwise having to trust the SE Root CA. It also anonymizes the static device information to external parties. During the owner pairing, [E] is transferred to the vehicle and is used to verify the Digital Key Certificate [H].

The certificate should have a short lifetime and should be renewed often. Revocation may be achieved by stopping the renewal process or through revocation methods (CRL, OCSP), which are out of scope of this specification.

16.2.6 [F] – Device OEM CA Certificate (Signed by Vehicle OEM)

The Vehicle OEM CA public key is embedded in the Vehicle OEM CA Certificate [J] in the vehicle (for owner pairing) and in authorized PK in the owner SE (for key sharing). This Vehicle OEM CA public key is used to verify the Vehicle OEM signed Device OEM CA Certificate [F] (note that [F] is also referred to as an “external CA certificate” in Section 15). [F] is then used to verify the Device OEM signed Instance CA Certificate [E]. [F] shall be stored in the device OS of all devices eligible for Digital Keys.

16.2.7 [G] – Digital Key per vehicle

[G] is the Digital Key generated by the Digital Key applet. There is one Digital Key per vehicle. During owner pairing, all Digital Key elements are provided by the vehicle and transferred to the device.

16.2.8 [H] – Digital Key Certificate

When a Digital Key is created, the applet signs the public key of the Digital Key structure using the corresponding Instance CA’s private key (see Section 4.1) to create the Digital Key Certificate [H]. The certificate is required for owner pairing and key sharing.

16.2.9 [J] – Vehicle OEM CA Certificate

This certificate is the Vehicle OEM root of trust for the Digital Key system. [J] is trusted by all Device OEMs and the corresponding Vehicle OEM. The private key corresponding to the Vehicle OEM CA Certificate is used to sign the Device OEM CA Certificate [D] to become [F]. [F] is transferred to the vehicle in order to verify the certificate chain at owner pairing, in which [F] is first verified by [J]. This leads to the Digital Key attestation as shown in Figure 6-7.

The public key of this certificate is also embedded in the owner’s Digital Key (as authorized PK). It is used during key sharing to verify the certificate chain, i.e., to validate the receiver’s public key origin.

The Device OEM may embed [J] on the device and use it to validate the authenticity of the vehicle public key (i.e., Vehicle.PK) contained in [K].

16.2.10 [K] – Vehicle Public Key Certificate

Before accepting the Digital Key Creation Data [L] from the vehicle during owner pairing, the device verifies the origin of the vehicle public key (i.e., Vehicle.PK) using the Vehicle Public Key Certificate [K]. The Vehicle.PK is attested to by the Vehicle OEM CA Certificate [J].

Note: The device uses the Vehicle OEM CA Certificate [J] or Device OEM signed Vehicle OEM CA Certificate [M] to verify [K] as described in Sections [16.2.9](#) and [16.2.12](#), respectively, before using [K] to verify Digital Key Creation Data [L].

16.2.11 [L] – Digital Key Creation Data

The Digital Key Creation Data [L] (not shown in Figure 16-1) is described in Figure 6-5.

The authorized public keys are the root of trust for key sharing.

Authorized Public Key #1: Vehicle OEM CA PK (mandatory)

This is used as the root for the key sharing validation chain. It cannot be updated once the Digital Key is created.

16.2.12 [M] – Vehicle OEM CA Certificate (signed by Device OEM)

The device may use the Device OEM signed Vehicle OEM CA Certificate [M] to validate the authenticity of the vehicle public key (i.e., Vehicle.PK) contained in the Vehicle Public Key Certificate [K]. The root of the validation chain is the Device OEM CA Certificate [D], which is trusted by the device. The public key of the Device OEM is used to validate the signature of [M].

16.2.13 [N] – Instance CA Attestation (self-signed) per Vehicle OEM

This attestation applies to Variant 2.

[N] is generated when a new Instance CA is created in the Digital Key applet. [N] contains the public key of the Instance CA in the Digital Key applet and is self-signed by the Instance CA using its private key, instanceCA.SK.

[N] is retrieved and verified by the Device OEM Server, and then the Device OEM Server issues a new Instance CA Certificate [E] by signing the InstanceCA.PK with the DeviceOEM.SK.

When implementing Variant 2, the SE root of trust is ensured by the following mechanisms:

- The Digital Key applet generates the Instance CA Attestation [N].
- The Device OEM Server opens a secure channel with the Digital Key applet to retrieve [N].
- One of the SCPs defined by GlobalPlatform (e.g., SCP03) shall be used. This secure channel shall be set to provide, at a minimum, integrity and data origin authentication on the APDU responses (i.e., requesting R-MAC with or without R-ENCRYPTION in parameter “i” when SCP03 is used).
- The Digital Key applet relies on its security domain for the implementation of the secure channel.
- The secure channel keys used by the security domain (i.e., AES keys when SCP03 is used) are specific to a given SE and are trusted by the Device OEM Server.

- When the Device OEM Server analyzes the response of the Digital Key applet, a successful verification of the R-MAC ensures that the data received over this secure channel has been generated by the expected Digital Key applet in the SE.

Figure 16-2 depicts the adaptation of the certificate chain model for Variant 2.

After successful verification of the Instance CA Attestation [N], the Instance CA Certificate [E] is issued by the Device OEM Server.

16.2.14 Server Certificate Relationships

Figure 16-3: SBxD/KIS Certificate chain

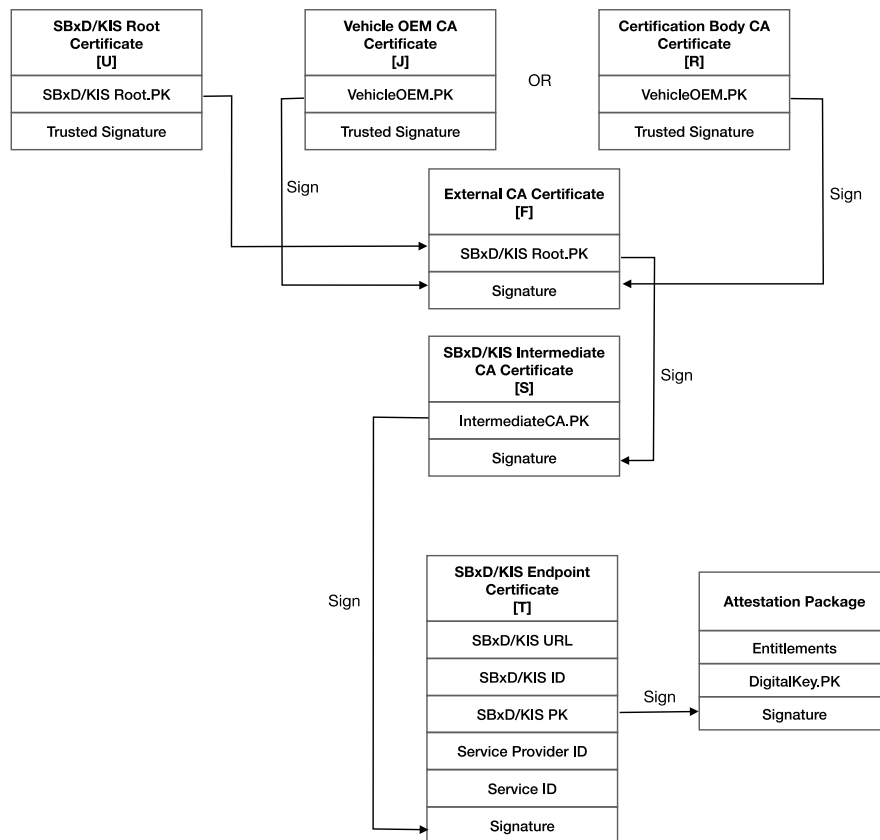


Figure 16-3 provides an overview of the server certificate relationships for server-issued keys.

16.2.14.1 SBxD/KIS endpoint certificate [T]

The SBxD/KIS endpoint certificate [T] attests to the key server identity and contains an additional extension with all required information to identify the service that is delivered by the server. The following requirements are applicable for [T]

- One key server can deliver multiple services for different service providers. Also, a service of a service provider can be split over multiple key servers.
- A dedicated SBxD/KIS endpoint certificate shall be used for each key server provider/service provider/service combination.

- A dedicated service identifier and key server provider identifier are part of the SBxD/KIS endpoint certificate. Key server provider identifiers are registered with CCC and can be unambiguously identified. Service providers do not need to be registered with CCC, the definition of the identifier is outside of the Digital Key specification scope.

16.2.14.2 SBxD/KIS Intermediate CA certificate [S]

The SBxD/KIS endpoint certificate [T] might be issued by an SBxD/KIS intermediate CA [S], if it is necessary for the PKI deployment (as shown in [Figure 16-3](#)). The intermediate CA (or the key server endpoint certificate [T] directly) are signed by the SBxD/KIS root CA [U], which is cross-signed by either the vehicle OEM root CA [J] or if supported by the vehicle OEM, a CCC-approved certification body root CA [R]. Each root CA or external CA certificate will be embedded in all mobile devices and SBFD key servers to be able to verify the sender's certificate chain.

16.3 Certificate Size Restrictions

This section defines X.509 certificate size restrictions to ensure compatibility with embedded system (memory) constraints. Maximum certificate sizes are chosen to allow the addition of proprietary, non-critical extensions in future certificate versions.

Vehicle OEM CA Certificate [J]

The maximum size of the mandatory elements, see Appendix C, of the DER-encoded certificate as per RFC 5280 [\[3\]](#) is 450 bytes.

The length of commonName (CN) fields for subject and issuer is limited to 30 bytes, independently of the chosen value type.

The Vehicle OEM can increase the size of the certificate by adding additional elements to extensions sequence as per RFC 5280 [\[3\]](#) up to a maximum size of an additional 350 bytes.

Vehicle OEM Intermediate Certificate

The maximum size of the mandatory elements, see Appendix C, of the DER-encoded certificate as per RFC 5280 [\[3\]](#) is 450 bytes.

The length of commonName (CN) fields for subject and issuer is limited to 30 bytes, independently of the chosen value type.

The Vehicle OEM can increase the size of the certificate by adding additional elements to extensions sequence as per RFC 5280 [\[3\]](#) up to a maximum size of an additional 350 bytes.

Vehicle public key certificate [K]

The maximum size of the mandatory elements, see Appendix C, of the DER-encoded certificate as per RFC 5280 [\[3\]](#) is 450 bytes.

The Vehicle OEM can increase the size of the certificate by adding additional elements to extensions sequence as per RFC 5280 [\[3\]](#) up to a maximum size of an additional 350 bytes.

External CA Certificate, issued by Vehicle OEM

The maximum size of the DER-encoded certificate as per RFC 5280 [\[3\]](#) is 650 bytes.

The length of commonName (CN) fields for subject and issuer is limited to 30 bytes, independently of the chosen value type.

Instance CA Certificate

The maximum size of the DER-encoded certificate as per RFC 5280 [3] is 650 bytes.

The length of commonName (CN) fields for issuer is limited to 30 bytes independently of the chosen value type.

Note: The subject CN field size is limited by the TLV definition in section “endpoint configuration” of the applet description (see Table 15-13).

Endpoint Certificate

The certificate shall include all authorized public keys that the vehicle provides.

In this version of the specification, the owner endpoint certificate shall be created with exactly one authorized public key (enforced by car); the receiver endpoint certificate shall be created with exactly zero authorized public keys (enforced by owner phone DK framework).

The applet allows for up to five authorized public keys per endpoint; usage of more than one authorized public key at the secure element level is kept open for future use cases.

The maximum size, of the DER-encoded certificate as per RFC5280 [3], shall be 650 bytes when holding one authorized public key. The maximum size shall be 970 bytes when holding five authorized public keys.

Note: The subject CN field size is limited by the TLV definition in section “endpoint configuration” of the applet description (see [Table 15-13](#)).

Encryption Key Attestation

The size of this attestation (not an X.509 certificate) is invariable.

16.4 Device Certificate Chain

The root of trust is moved from the SE Root to the Device OEM. The Device OEM then trusts and attests to signatures done by the on-device Instance CA.

→ attest/verify

⇒ trust

Instance CA chains:

SE Root CA Certificate → Instance CA Attestation

Device OEM CA ⇒ SE Root CA

Device OEM CA ⇒ Instance CA

Device OEM CA Certificate → Instance CA Certificate

16.5 Vehicle Certificate Chain

The vehicle public key is attested to by a Vehicle OEM CA Certificate in the device OS. This certificate is trusted by the Device OEM.

→ attest /verify

⇒ trust

Device OEM CA ⇒ Vehicle OEM CA

Device OEM embeds Vehicle OEM CA Certificate in device

Vehicle OEM CA Certificate → Vehicle Public Key Certificate

16.6 Supported Verification Chain

The following chain shall be supported by Digital Key applet for key sharing:

Authorized PK → External CA certificate → Instance CA Certificate → endpoint

16.7 Owner Pairing Certificate Chain

At owner pairing, the vehicle verifies the authenticity and the correctness of the key created in the SE. The key properties, including the device public key, are attested to by the Digital Key certification chain, which is rooted in the vehicle.

16.7.1 Device and Vehicle

→ attest /verify

⇒ trust

Vehicle OEM CA ⇒ Device OEM CA

Device OEM CA ⇒ Instance CA Certificate

Vehicle OEM CA Certificate → Device OEM CA Certificate → Instance CA Certificate → Digital Key Certificate

16.8 Key Sharing Certificate Chain

Before issuing the Digital Key to the receiver, the owner device verifies the authenticity of the public key created in the receiver device. The receiver device may be a different device brand than the owner device.

It is assumed that the vehicle has provided to the owner SE the Vehicle OEM CA PK, stored safely in the authorized_PK structure within the Digital Key Creation Data [L]. For verification, the receiver device provides the Instance CA Certificate [E] signed by the Device OEM and the receiver device OEM CA Certificate [F] signed by the Vehicle OEM.

→ attest /verify

⇒ trust

The owner device verifies:

Vehicle OEM CA PK → receiver device OEM CA Certificate → receiver Instance CA Certificate → receiver Digital Key Certificate

- 1 The owner device signs the receiver Digital Key Certificate using the owner's private key. The
- 2 vehicle verifies the owner signature before accepting the receiver's public key when presented
- 3 by the receiver device or online through the Vehicle OEM Server:
- 4 Owner PK → receiver Digital Key Certificate

17 SERVER-TO-SERVER COMMUNICATIONS AND API

17.1 Introduction

This section describes the API specification for communications between

- Device OEM Servers and Vehicle OEM Servers that support domain version DS-VS $\geq$ 3.0.
- SBOD/SBFD and FMS/Service Provider
- SBOD/SBFD and Vehicle OEM Server

Request and response bodies shall be formatted as JSON. The communication protocol used for all interfaces shall be HTTPS. All strings shall be UTF-8 encoded (Unicode Normalization Form C (NFC)).

For documentation purposes, the API is documented using YAML format as outlined by the OpenAPI Specification [<Insert reference to OpenAPI specification>].

Non-mandatory parameters are considered as optional or conditional. Conditions under which conditional parameters shall be included are documented in the “Description” column of each API description table.

17.2 API Design

- All APIs shall be implemented as HTTPS POST requests unless otherwise specified.
- Request and response payloads shall consist of simple JSON based data structures that allow for future expansion. The main data types used are array, dictionary, boolean, and string.
 - If a mandatory parameter is absent in the request payload carried in an API, then the receiver of the API shall return HTTP status code 422 with sub-status error code 50136 (see [Table 17-68](#)).
 - If a mandatory parameter is absent in the response payload carried in an API, then the receiver of the API shall ignore and discard the complete response payload.
 - If an unknown parameter is present in the request or response payload carried in an API, then the receiver of the API shall ignore and discard the unknown parameter and proceed processing the response payload
- Error codes are specified at three levels:
 - Transport level error codes and Detailed error codes are described in Section [17.11.3](#)
 - HTTP error codes are described in [Table 17-67](#)
 - Optional Sub Status code field in common response header as described in [Table 17-68](#)
 - Standardized application layer error codes
 - Status code field in common response header; values defined in Section [17.11.3](#).

17.3 Security

Client application shall be hosted on user's devices. Device OEM Server and Vehicle OEM Server shall be hosted in secure data centers.

Server APIs shall be supported only over https. Sensitive data elements, where applicable, shall be protected with additional encryption protocols. Server APIs shall be supported only over https with mutual authentication, i.e., 2-way TLS.

17.4 Versions

Server APIs send and receive data that can originate from another server or from a device connected to the other server (if other server is device OEM server).

Domain Version in the description of the APIs indicate the domain version that a data element depends on. The negotiation of the API version used between the Device OEM Server and the Vehicle OEM Server (DS-VS) is outside of this specification. Versions are 2-byte values as described in Section 2.10.5. The MSB indicates the major version. The LSB indicates the minor version. All version lists shall be sorted in descending order. For example, vodfwVehicleList = ["3.0", "1.0"].

17.5 URL Scheme

All server services URIs shall conform to the following scheme:

scheme://host:port/{service}/{version}/apiName

where:

• **scheme** = https

• **host:port** = Host name and Port number for the endpoint.

• **service** = Target application that hosts the API endpoint.

• **version** = Version number of the schema. Current major version is v3.

• **apiName** = Name of the API being called

An API version shall be included in the URI for all interfaces. The first supported version is v1. The version shall be incremented by 1 for major API changes or backward incompatible iterations on existing APIs.

17.6 User Interface

17.6.1 Description

User interfaces are created and controlled by Device OEM entities. Each Device OEM can customize the user experience as per its platform.

The Vehicle OEM provides all necessary visual elements for a specific vehicle/model to the Device OEM in an offline process, prior to going live. Details of the visual elements are out of scope of the specification and should be negotiated between the Vehicle OEMs and Device OEMs. An identifier, which can be chosen by the Vehicle OEM, is assigned for UI elements. This identifier is called uiIdentifier.

To ensure uniqueness across Vehicle OEMs, a prefix may be used. The uiIdentifier should be generic and the system should not be able to identify owner or receiver using uiIdentifier. An example of an uiIdentifier for a specific vehicle: “VehicleOEMNameModelYearColor.” The UI elements are stored on the Device OEM Server to ensure the best availability at runtime. During different types of provisioning (owner pairing, key sharing), the uiIdentifier is provided to the Device OEM Server by the Vehicle OEM Server. The Device OEM may then select the predefined UI elements. It is the responsibility of the Device OEM to display the UI elements appropriate to device-specific parameters.

17.7 APIs implemented by Vehicle OEM Server

17.7.1 API – migrationMetadata

This API endpoint is designed for handling migration processes on the device as described in Section [2.10.5](#)

17.7.1.1 End Point

HTTPS POST /{service}/{version}/migrationMetadata

17.7.1.2 Request

Table 17-1: migrationMetadata() Request

Parameter	Required (M/O/C)	Type	Description	Domain Version
keyID	M	string	The unique identifier for a key. Computed as the 160-bit SHA-1 hash of the value of the bit string subjectPublicKey from the keyData (excluding the tag, length, and number of unused bits).	V-OD-FW
vodfwCurrent	M	string	The current version of V-OD-FW chosen by the device. This field is provided when the V-OD-FW version supported by the device is being updated.	
vodfwTarget	M	string	The target version of V-OD-FW chosen by the device. This field is provided when the V-OD-FW version supported by the device is being updated. vodfwTarget shall be ≥ 3.0 .	
dvsCurrent	M	string	The current version of D-VS chosen by the device. This field is provided when the D-VS version supported by the device is being updated	
dvsTarget	M	string	The target version of D-VS chosen by the device. This field is provided when the D-VS version supported by the device is being updated.	

17.7.1.3 Response

Table 17-2: migrationMetadata() Response: Status 200: OK

Parameter	Required (M/O/C)	Type	Description	Domain Version
responseHeader	M	Object: ResponseHeader	ResponseHeader with status Code = 200: OK. See Table 17-36	N/A
migrationMetadata	C	Object: EncryptedDataContainer	Absent when no metadata requires upgrade. Payload (data field of EncryptedDataContainer (Table 17-37)) contains UnencryptedMigrationMetadata (Table 17-45) which is used to convert existing Digital Keys to Sharing in a Chain capable keys.	N/A

17.7.1.4 Samples

Listing 17-1: Sample MigrationMetadata Request Body

```
{
  "keyID": "2A2DA3920A83CC97B35E338AB72898A2A9776485",
  "vodfwCurrent": "1.0",
  "vodfwTarget": "3.0",
  "dvsCurrent": "1.0",
  "dvsTarget": "3.0"
}
```

Listing 17-2: Sample MigrationMetadata Response

```
{
  "responseHeader": {
    "statusCode": 200
  },
  "migrationMetadata": {
    "version": "ECIES_v1",
    "ephemeralPublicKey":
"04613197827d91806d630bc4adff44686b012316eb03825f2d6587ffd58d32f4522ada80cc
93679e1a316dc0729ebf8172fd41f0c0c1bdda01126f1a6186b2a008",
    "publicKeyHash": "8C4d7Cf4312CB24e4e95fddD",
    "data": "kE8/Rs9sNo4olc6s3Qej7mXhAtDPcp ...1Bm/r7bKNX6m365vnIDEdc"
  }
}
```

17.7.2 API - recoverKeyData

This is a generic API offered by the Vehicle OEM Server to retrieve metadata of active keys. This API should only be called if the device loses some of the information listed in this section and shall not be used for regular sync. Vehicle OEM specific request policy may apply (out of scope of this CR). It is recommended that device OEM Server should store metadata of active keys to avoid calling the API in order to reduce the load on Vehicle OEM Server.

Attention: key metadata returned using this API might differ from what was known to the device OEM at the time of trackKey. If metadata returned is indeed different, the device OEM shall use the returned values from then on.

17.7.2.1 End Point

HTTPS POST /{service}/{version}/recoverKeyData

17.7.2.2 Request

Table 17-3: recoverKeyData() Request

Parameter	Required (M/O/C)	Type	Description	Domain Version
keyID	M	string	The unique identifier for a key. Computed as the 160-bit SHA-1 hash of the value of the bit string subjectPublicKey from the keyData (excluding the tag, length, and number of unused bits).	V-OD-FW

17.7.2.3 Response

Table 17-4: recoverKeyData() Response: Status Code 200: OK

Parameter	Required (M/O/C)	Type	Description	Domain Version
responseHeader	M	Object: ResponseHeader	ResponseHeader with status Code = 200: OK. See Table 17-36	
status	M	Object: KeyStatus	Status of the Digital Key. See Table 17-56	
uiBundle	M	Object: UiBundle	See Table 17-46	
vodfwVehicleList	C	List	Only required by owner device. Contains in descending order the vehicle owner device framework version V-OD-FW vehicle list, with V-OD-FW agreed Version first. This list is initially exchanged in Tag 5B _h include in SPAKE2+ request (Table 5-4) or Key Creation Request (Table 11-5).	V-OD-FW
dvsVehicleServer List	C	List	Shall be included if the responding entity is the vehicle OEM server. Contains the sorted list in descending order of all versions the	D-VS

Parameter	Required (M/O/C)	Type	Description	Domain Version
			vehicle OEM server supports for communication with the device (D-VS), with D-VS agreed version first.	
encryptedDeviceData	M	Object: EncryptedDataContainer	Sensitive key information encrypted from the Vehicle OEM Server to device using Device.Enc.PK as defined in Section 14. Payload (data field of EncryptedDataContainer (Table 17-37)) contains unencryptedDeviceData (Table 17-39)	
instanceFreshness	O	int32	Instance CA freshness check of Owner/sender Device	

17.7.2.4 Samples

Listing 17-3: Sample reccoverKeyDataRequest Body

```
1. {
2.   "keyID": "6A6C8BFACEF355635DF1E5BFDF805E90F6CBB703"
3. }
```

Listing 17-4: Sample recoverKeyData() Response

```
1. {
2.   "responseHeader": {
3.     "statusCode": 200
4.   },
5.   "status": "ACTIVE",
6.   "uiBundle": {
7.     "keyInfo": {
8.       "groupIdentifier": "3f66",
9.       "entitlement": {
10.        "accessProfile": 0,
11.        "accountRole": "88CF"
12.      },
13.       "validFrom": "2024-01-17T16:22:05.677Z",
14.       "validTo": "2024-01-17T16:22:05.677Z"
15.     },
16.     "vehicleInfo": {
17.       "uiIdentifier": "8dcd8010-d3ac-4f24-bd94-de144e93acaf",
```



```
18.     "brand": "xyz",
19.     "model": "abc"
20. },
21.   "sharingInfo": {
22.     "activationRequired": true,
23.     "supportedEntitlements": [
24.       "entitlement": {
25.         "accessProfile": 0,
26.         "accountRole": "88CF"
27.       }
28.     ],
29.     "shareableKeysCount": 3,
30.     "sharedKeysCount": 1
31.   }
32. },
33.   "vodfwVehicleList": [
34.     "3.0",
35.     "1.0"
36.   ],
37.   "dvsVehicleServerList": [
38.     "3.0",
39.     "1.0"
40.   ],
41.   "encryptedDeviceData": {
42.     "version": "ECIES_v1",
43.     "ephemeralPublicKey":
44.     "04613197827d91806d630bc4adff44686b012316eb03825f2d6587ffd58d32f4522ada80cc93679e
45.     1a316dc0729ebf8172fd41f0c0c1bdda01126f1a6186b2a008",
46.     "publicKeyHash":
47.     "D52a21E0E1C55712C17D8Bcd738B8e603eE21cF35AfB70E9CE9ee4b5F6530aFc1AaF3f55AC0f7ABD
48.     B52",
49.     "data": "kE8/Rs9sNo4olc6s3Qej7mXhAtDPcp ...1Bm/r7bKNX6m365vnIDEDc"
50.   },
51.   "instanceFreshness": 255
52. }
```

Listing 17-5: Unsuccessful Sample recoverKeyData() Response

```
1. {
2.   "statusCode": 404,
3.   "subStatusCode": 50110,
4.   "statusMessage": "Unknown key id supplied"
5. }
```

17.7.3 API - trackKey

This is a generic API offered by the Vehicle OEM Server to track a Digital Key. This API should have idempotent behavior. Idempotency in this case means that if trackKey() Request is called for a keyID that is already registered, the API call shall be successful and return with the same response as the response for the initial trackKey() Request. This API is used in owner pairing and key sharing as described in Section 6.3.4.3 and Section 11.4, respectively.

17.7.3.1 End Point

HTTPS POST /{service}/{version}/trackKey

17.7.3.2 Request

Table 17-5: trackKey() Request Body

Parameter	Required (M/O/C)	Type	Description	Domain Version
keyID	M	string	The unique identifier for a key. Computed as the 160-bit SHA-1 hash of the value of the bit string subjectPublicKey from the keyData (excluding the tag, length, and number of unused bits).	V-OD-FW
deviceType	M	string	See Table 17-57	DS-VS
keyOrigin	M	string	See Table 17-58	DS-VS
accountIdHash	C	string	accountIdHash is defined in [41]. In trackKey() sent during Partial Migration, accountIdHash shall be present. Also required by vehicle OEM server to generate the same group identifier for V1 and V3 keys. Otherwise, optional. Facilitates vehicle OEM server in Digital Key management across different Digital Key versions, belonging to the same Device OEM account identifier.	D-VS
vodfwAgreedVersion	M	string	The agreed-upon V-OD-FW domain version.	D-VS

Parameter	Required (M/O/C)	Type	Description	Domain Version
dvsAgreedVersion	M	string	The agreed-upon D-VS domain version.	D-VS
encryptedKeyData	C	Object: EncryptedDataContainer	Encrypted from Device to Vehicle OEM Server. Encrypted using VehicleOEM.Enc.PK as defined in Section 14. Refer to Section 6.3.4.4 (see Table 6-2) and Section 11.6 (see Table 11-17). Conditional: Not present when trackKey is called during Partial Migration.	Encryption : D-VS Data: V-OD-FW, D-VS

1

2 17.7.3.3 Response

3

Table 17-6: trackKey() Response: Status Code 200: OK

Parameter	Required (M/O/C)	Type	Description	Domain Version
responseHeader	M	Object: ResponseHeader	ResponseHeader with status Code = 200: OK. See Table 17-36	
uiBundle	M	Object: UiBundle	See Table 17-46	DS-VS
status	M	Object: KeyStatus	See Table 17-56	DS-VS
encryptedDeviceData	C	Object: EncryptedDataContainer	Absent if trackKey is called after partial migration. Payload (data field of EncryptedDataContainer (Table 17-37)) contains unencryptedDeviceData (Table 17-39)	Encryption : D-VS Data: V-OD-FW, D-VS
vodfwVehicleList	M	List	Contains in descending order the vehicle owner device framework version V-OD-FW list, with V-OD-FW-agreedVersion first. This list is in initially exchanged in Tag 5B _h include in SPAKE2+ request (Table 5-4) or Key Creation Request (Table 11-5).	

4 17.7.3.4 Samples

5

Listing 17-6: Sample trackKey() Request Body

```

1. {
2.   "keyID": "F7E388FBC842D9CB1C8973AEAABEFBC80D5D8C9C",
3.   "deviceType": "PHONE",

```

```
4.    "keyOrigin": "OWNER_PAIRING",
5.    "accountIdHash": "C8645830202EFEB53427A6D75F15C85E78A5195307E2351858349AB9",
6.    "vodfwAgreedVersion": "3.0",
7.    "dvsAgreedVersion": "3.0",
8.    "encryptedKeyData": {
9.        "version": "ECIES_v1",
10.        "ephemeralPublicKey":
11.        "04613197827d91806d630bc4adff44686b012316eb03825f2d6587ffd58d32f4522ada80cc93679
12.        e1a316dc0729ebf8172fd41f0c0c1bdda01126f1a6186b2a008",
13.        "publicKeyHash": "DAf2237499DBd059E957230abFAF46Bf0fdD56AF",
14.        "data": "kE8/Rs9sNo4olc6s3Qej7mXhAtDPcp ...1Bm/r7bKNX6m365vnIDEDc"
15.    }
16. }
```

1

2

Listing 17-7: Sample trackKey() Response: Status: 200 OK.

```
1.  {
2.    "responseHeader": {
3.        "statusCode": 200
4.    },
5.    "uiBundle": {
6.        "keyInfo": {
7.            "groupIdentifier": "3f66",
8.            "entitlement": {
9.                "accessProfile": 0,
10.            "accountRole": "88CF"
11.        },
12.        "validFrom": "2024-01-17T16:22:05.677Z",
13.        "validTo": "2024-01-17T16:22:05.677Z"
14.    },
15.    "vehicleInfo": {
16.        "uiIdentifier": "8dcd8010-d3ac-4f24-bd94-de144e93acaf",
17.        "brand": "xyz",
18.        "model": "abc"
19.    },
20.    "sharingInfo": {
21.        "activationRequired": true,
22.        "supportedEntitlements": [
23.            "entitlement": {
24.                "accessProfile": 0,
```

```
25.         "accountRole": "88CF"
26.     }
27. },
28.     "shareableKeysCount": 3,
29.     "sharedKeysCount": 1
30. }
31. },
32. "status": "ACTIVE",
33. "encryptedDeviceData": {
34.     "version": "ECIES_v1",
35.     "ephemeralPublicKey":
"04613197827d91806d630bc4adff44686b012316eb03825f2d6587ffd58d32f4522ada80cc93679e
1a316dc0729ebf8172fd41f0c0c1bdda01126f1a6186b2a008",
36.     "publicKeyHash": "Fde27Bc80E1F58CFD0e4eE0F84a71F1EccA",
37.     "data": "kE8/Rs9sNo4olc6s3Qej7mXhAtDPcp ...1Bm/r7bKNX6m365vnIDEDc"
38. },
39. "vodfwVehicleList": [
40.     "3.0",
41.     "1.0"
42. ],
43. "dvsVehicleServerList": [
44.     "3.0",
45.     "1.0"
46. ]
47. }
```

1

2 17.7.4 API – *inFleet*

3 This is a generic API offered by vehicle OEM server for **SBOD** to request the infleeting of a new
4 vehicle.

5 17.7.4.1 Endpoint

6 HTTPS POST /{service}/{version}/inFleet

17.7.4.2 Request

Table 17-7: inFleet() Request body

Parameter	Required (M/O/C)	Type	Description	Domain Version
vehicleId	M	string	The unique identifier of the vehicle. This should be the VIN of the vehicle, but any unique identifier may also be used.	D-VS
vehicleOEMIdentifier	C	string	Vehicle OEM Identifier as listed in [35]. Included if vehicleId is not the VIN of the vehicle.	
proofOfOwnership	O	string	Proof that the calling entity owns the vehicle	

17.7.4.3 Response

Table 17-8: inFleet() Response body: Status 200: OK

Parameter	Required (M/O/C)	Type	Description	Domain Version
responseHeader	M	Object: Response Header	ResponseHeader with status Code = 200: OK. See Table 17-36	N/A
vehicleCertificateChain	M	array	Base 64 encoded array of X509 certificates in DER format representing a certificate chain. See Section 16.5	
endpointCertificateExtensionData	M	string	Endpoint Certificate Extension Data. See Section 12.2.2.2 and Listing 15-4	
vodfwVehicleList	O	list	Contains in descending order the vehicle owner device framework version V-OD-FW vehicle list, with V-OD-FW agreed version first. This list is initially exchanged in Tag 5B _h included in SPAKE2+ request (Table 5-4) or Key Creation Request (Table 11-5).	
dvsVehicleServerList	O	list	Contains the sorted list in descending order of all version the Vehicle OEM Server supports for communication with the Device (D-VS), with D-VS agreed version first.	

17.7.5 API – registerKey

This is a generic API offered by vehicle OEM server for SBOD to send the SBOD signing key used in the vehicle.

17.7.5.1 End Point

HTTPS POST /{service}/{version}/registerKey()

17.7.5.2 Request

Table 17-9: registerKey() Request body

Parameter	Required (M/O/C)	Type	Description	Domain Version
vehicleId	M	String	The unique identifier of the vehicle. This should be the VIN of the vehicle, but any unique identifier may also be used.	
sbodEndpointCertificate Chain	M	List of Strings	Base 64 encoded array of X509 certificates in DER format representing a certificate chain for SBOD endpoint certificate. See Section 16.2.14	

17.7.5.3 Response

Table 17-10: registerKey() Response body: Status 200: OK

Parameter	Required (M/O/C)	Type	Description	Domain Version
responseHeader	M	Object: ResponseHeader	ResponseHeader with status Code = 200: OK. See Table 17-36	N/A
sharingInfo	M	Object: SharingInfo	Object that contains all UiBundle data related to sharing of the key. Response object when the signing key of the SBOD was successfully sent to the vehicle. See Table 17-48	

17.7.6 API – preShare

This is a generic API offered by the Vehicle OEM Server to submit data needed for subsequent key sharing. This API should have idempotent behavior. If preShare is called for a keyID that is already registered, the API call shall be successful and return with a usual response. This API is used in secure key sharing as described in Section [11.5.2](#).

17.7.6.1 End Point

HTTPS POST /{service}/{version}/preShare

17.7.6.2 Request

Table 17-11: preShare() Request Body

Parameter	Required (M/O/C)	Type	Description	Domain Version
keyID	M	string	Sender Key Identifier. The unique identifier for a key. Computed as the 160-bit SHA-1 hash of the value of the bit string subjectPublicKey from the keyData (excluding the tag, length, and number of unused bits).	
preShareDataType	M	string	See Table 17-65	
preShareDataArray	M	List of Objects of type EncryptedDataContainer (Table 17-37)	Encrypted from Device to Vehicle OEM Server. Encrypted using VehicleOEM.Enc.PK as defined in Section 14. Payload (data field of EncryptedDataContainer (Table 17-37)) contains signedSharingSecInfo Tag 7F41 _h (Table 11-23)	D-VS ?

17.7.6.3 Response

Table 17-12: preShareResponse Body: Status 200: OK

Parameter	Required (M/O/C)	Type	Description	Domain Version
responseHeader	M	Object: ResponseHeader	ResponseHeader with status Code = 200: OK See Table 17-36	N/A

17.7.6.4 Samples

Listing 17-8: Sample preShare() Request

```

1.  "keyID": "F7E388FBC842D9CB1C8973AEAABEFBC80D5D8C9C",
2.  "preShareDataType": "SENDER_ONLINE_SHARING_PIN",
3.  "preShareDataArray": [{
4.    "version": "ECIES_v1",
5.    "ephemeralPublicKey":
    "04613197827d91806d630bc4adff44686b012316eb03825f2d6587ffd58d32f4522ada80cc93679
    e1a316dc0729ebf8172fd41f0c0c1bdda01126f1a6186b2a008",

```



```

6.     "publicKeyHash": "Fde27Bc80E1F58CFD0e4eE0F84a71F1EccA",
7.     "data": "kE8/Rs9sNo4o1c6s3Qej7mXhAtDPcp ...1Bm/r7bKNX6m365vnIDEDc"
8.   }]
```

17.7.7 API – preTrack

This is a generic API offered by the Vehicle OEM Server to submit data needed for subsequent key sharing. This API should have idempotent behavior. If preTrack is called for a keyID that is already registered, the API call shall be successful and return with a usual response. This API is used in key sharing as described in Section [11.5.3](#).

17.7.7.1 End Point

HTTPS POST /{service}/{version}/preTrack ()

17.7.7.2 Request

Table 17-13: preTrack() Request Body

Parameter	Required (M/O/C)	Type	Description	Domain Version
keyID	C	string	Receiver Key Identifier. Included if preTrackDataType = UI_BUNDLE. The unique identifier for a key. Computed as the 160-bit SHA-1 hash of the value of the bit string subjectPublicKey from the keyData (excluding the tag, length, and number of unused bits).	
preTrackDataType	M	Enum (max length 32 bytes string)	Type of content delivered in vehicleData. See Table 17-64	DS-VS
vehicleData	M	Object: EncryptedDataContainer (Table 17-37)	Encrypted from Device to Vehicle OEM Server. Encrypted using VehicleOEM.Enc.PK as defined in Section 14. If preTrackDataType = ONLINE_SHARING_PIN, then Encrypted data = Unencrypted Signed Online Sharing PIN Information Package Tag 7F43_h including outer tag 7F43_h (see Table 11-15) If preTrackDataType = UI_BUNDLE, then Encrypted data = unencrypted tag 4D_h (vehicle_identifier) including outer tag 4D_h	Data: D-VS

17.7.7.3 Response

Table 17-14: *preTrack()* Response Body: Status 200: OK

Parameter	Required (M/O/C)	Type	Description	Domain Version
responseHeader	M	Object: ResponseHeader	ResponseHeader with status Code = 200: OK. See Table 17-36	N/A
vehicleInfo	C	Object: vehicleInfo (Table 17-55)	Absent, if Online Sharing pin verification fails.	DS-VS
attemptsLeft	C	Integer	Only present if PIN validation failed. It indicates number of PIN entry attempts left. Minimum value = 0 Maximum value = 9	DS-VS

17.7.7.4 Samples

Listing 17-9: *preTrack()* Request

```
1.  "keyID": "F7E388FBC842D9CB1C8973AEAABEFBC80D5D8C9C",
2.  "preTrackDataType": "ONLINE_SHARING_PIN",
3.  "vehicleData": {
4.    "version": "ECIES_v1",
5.    "ephemeralPublicKey":
    "04613197827d91806d630bc4adff44686b012316eb03825f2d6587ffd58d32f4522ada80cc93679
    e1a316dc0729ebf8172fd41f0c0c1bdda01126f1a6186b2a008",
6.    "publicKeyHash": "Fde27Bc80E1F58CFD0e4eE0F84a71F1EccA",
7.    "data": "kE8/Rs9sNo4olc6s3Qej7mXhAtDPcp ...1Bm/r7bKNX6m365vnIDEDc"
8.  }
```

```
1.  {
2.    "responseHeader": {
3.      "statusCode": 200
4.    },
5.    "vehicleInfo": {
6.      "uiIdentifier": "8dcd8010-d3ac-4f24-bd94-de144e93acaf",
7.      "brand": "xyz",
8.      "model": "abc"
9.    }
10. }
```

17.8 APIs implemented by Device OEM Server and Vehicle OEM Server

17.8.1 API - healthCheck

This is a generic API offered by the Device OEM Server and the Vehicle OEM Server to determine the availability of the corresponding server.

17.8.1.1 End Point

HTTPS GET /{service}/{version}/healthCheck

17.8.2 API - manageKey

This is a generic API offered by the Device OEM Server and Vehicle OEM Server to manage the lifecycle of a Digital Key. This API is used in key sharing and key termination as described in Section 11 and Section 13, respectively.

17.8.2.1 End Point

HTTPS POST /{service}/{version}/manageKey

17.8.2.2 Request

Table 17-15: manageKey() Request

Parameter	Required (M/O/C)	Type	Description	Domain Version
keyID	M	string	The unique identifier for a key. Computed as the 160-bit SHA-1 hash of the value of the bit string subjectPublicKey from the keyData (excluding the tag, length, and number of unused bits).	V-OD-FW
action	M	string	Action that needs to be performed by the receiving entity. See Table 17-59	DS-VS
terminationAttestation	O	string MaxLen = 4096 bytes	Termination attestation of a key when owner Digital Key or shared Digital Key is terminated on own device. For a detailed description of this field see Section 15.3.2.5 . Signed by DigitalKeyx.SK, as defined in Section 14 . Provided by sender/receiver device whenever possible	D-VS
deviceRemoteTerminationRequest	C	Object: EncryptedData Container	Remote termination request for termination of a remote key when shared key is terminated from device (can also be from a SBOD or Sbfd). For a detailed description of this field see Section 14.2 . Signed by	D-VS

Parameter	Required (M/O/C)	Type	Description	Domain Version
			DigitalKeyx.SK as defined in Section 14 . Provided by Owner Device. See Table 17-37	
serverRemoteTerminationRequest	C	Object: EncryptedData Container	Remote termination request for termination of a remote key when owner key or shared key is terminated from Vehicle OEM Server. For a detailed description of this field see Section 14.2 . Signed by VehicleOEM.Sig.SK as defined in Section 14 . See Table 17-37	D-VS
sbxdRemoteTerminationRequest	C	Object: EncryptedData Container	Remote termination request for termination of a remote key when owner key or shared key is terminated by an SBxD. See Table 17-37 and Table 17-52	
vehicleOEMProprietaryData	O	string MaxLen: 4096	Subsection of the private mailbox of a Digital Key containing the Vehicle OEM proprietary data when owner or shared key is terminated on the device (local or remote termination).	D-VS

1

2 17.8.2.3 Response

3

Table 17-16: manageKey() Response: Status Code 200: OK

Parameter	Required (M/O/C)	Type	Description	Domain Version
responseHeader	M	Object: ResponseHeader	ResponseHeader with status Code = 200: OK. See Table 17-36	
status	O	string	See Table 17-56	
encryptedDeviceData	C	Object: EncryptedDataContainer	Required when device data still remains after key management, e.g., last key was terminated, and no shared keys remain open. Contents of unencryptedDeviceData (Table 17-39) within EncryptedDataContainer (Table 17-37)	D-VS

17.8.2.4 Samples

Listing 17-10: Sample manageKey() Request Body

```
1. {
2.   "keyID": "F4F19630DFCF052F2F387AD54B2F6E710514149F",
3.   "action": "TERMINATE",
4.   "terminationAttestation":
5.     "5CFEB534D8F5CEFA6D1B780BCB5CFEB534D95912CEFA6D1B780BCB5CFE",
6.   "deviceRemoteTerminationRequest": {
7.     "version": "ECIES_v1",
8.     "ephemeralPublicKey":
9.       "04613197827d91806d630bc4adff44686b012316eb03825f2d6587ffd58d32f4522ada80cc93679e
10.      1a316dc0729ebf8172fd41f0c0c1bdda01126f1a6186b2a008",
11.     "publicKeyHash":
12.       "6a213DC65ADdAda4Cea8426e184cb9FfFAc8C10a039DC877cc73cb3e51EbC7FE51811541c0",
13.     "data": "kE8/Rs9sNo4olc6s3Qej7mXhAtDPcp ...1Bm/r7bKNX6m365vnIDEDc"
14.   },
15.   "vehicleOEMProprietaryData": "5CEFA6D1B780BCB5CFEB534D"
16. }
```

17.8.3 API – versionUpdate

This API is used to either inform about a processed change in domain versions or to retrieve the current domain versions supported by Device or Vehicle OEM. If an entity is looking to retrieve the peer domain versions, then it may include it’s own domain versions in the versionUpdate request.

17.8.3.1 End Point

HTTPS POST /{service}/{version}/versionUpdate

17.8.3.2 Request

Table 17-17: versionUpdate() Request

Parameter	Required (M/O/C)	Type	Description	Domain Version
keyID	M	string	The unique identifier for a key. Computed as the 160-bit SHA-1 hash of the value of the bit string subjectPublicKey from the keyData (excluding the tag, length, and number of unused bits).	V-OD-FW
vodfwDeviceList	C	List	Shall be included if calling entity is the device OEM. Contains in descending order the vehicle owner device framework version V-OD-FW device list. This list is in initially exchanged in Tag 5A _h include in SELECT response command (Table 5-3).	V-OD-FW

Parameter	Required (M/O/C)	Type	Description	Domain Version
vodfwVehicleList	C	List	Shall be included if the calling entity is the Vehicle OEM. Contains in descending order the vehicle owner device framework version V-OD-FW vehicle list, with V-OD-FW agreed Version first. This list is in initially exchanged in Tag 5B _h include in SPAKE2+ request (Table 5-4) or Key Creation Request (Table 11-5).	V-OD-FW
dvsDeviceList	C	List	Shall be included if the calling entity is the device OEM server. Contains the sorted list in descending order of all versions the device supports for communication with the vehicle OEM server (D-VS).	D-VS
dvsVehicleServer List	C	List	Shall be included if the calling entity is the vehicle OEM server. Contains the sorted list in descending order of all versions the vehicle OEM server supports for communication with the device (D-VS), D-VS agreed version first.	D-VS

1

2 17.8.3.3 Response

3

Table 17-18: versionUpdate() Response: Status Code 200: OK

Parameter	Required (M/O/C)	Type	Description	Domain Version
responseHeader	M	Object: ResponseHeader	ResponseHeader with status Code = 200: OK. See Table 17-36	N/A
vodfwDeviceList	C	List	Shall be included if calling entity is the device OEM. Contains in descending order the vehicle owner device framework version V-OD-FW device list. This list is in initially exchanged in Tag 5A _h include in SELECT response command (Table 5-3).	V-OD-FW
vodfwVehicleList	C	List	Shall be included if the calling entity is the Vehicle OEM. Contains in descending order the vehicle owner device framework version V-OD-FW vehicle list, with V-OD-FW agreed Version first. This list is in initially exchanged in Tag 5B _h include in SPAKE2+ request (Table 5-4) or Key Creation Request (Table 11-5).	V-OD-FW
dvsDeviceList	C	List	Shall be included if the responding entity is the device OEM server. Contains the sorted list in descending order of all versions the device supports for communication with the vehicle OEM server (D-VS).	D-VS

Parameter	Required (M/O/C)	Type	Description	Domain Version
dvsVehicleServerList	C	List	Shall be included if the responding entity is the vehicle OEM server. Contains the sorted list in descending order of all versions the vehicle OEM server supports for communication with the device (D-VS), with D-VS agreed version first.	D-VS

1 17.8.3.4 Samples

2 Listing 17-11: Sample versionUpdate() Request Body

```
1. {
2.   "keyID": "D95CD1A5C517D38960C6FA9FD76985CFDF530788",
3.   "vodfwDeviceList": [
4.     [
5.       "3.0",
6.       "1.0"
7.     ]
8.   ],
9.   "dvsDeviceList": [
10.    [
11.      "3.0",
12.      "1.0"
13.    ]
14.  ]
15. }
```

3 Listing 17-12: Sample versionUpdate() Response

```
1. {
2.   "responseHeader": {
3.     "statusCode": 200,
4.   },
5.   "vodfwVehicleList": [
6.     [
7.       "3.0",
8.       "1.0"
9.     ]
10.  ],
11.   "dvsVehicleServerList": [
12.     [
```

```

13.         "3.0",
14.         "1.0"
15.     ]
16. ]
17. }

```

17.9 APIs implemented by Device OEM Server

17.9.1 API – eventNotification

This is a generic API offered by the Device OEM Server and the Vehicle OEM Server to communicate different events on a Digital Key. Event notifications that are not supported by a receiving Device OEM server shall be accepted and ignored.

17.9.1.1 End Point

HTTPS POST /{service}/{version}/eventNotification

17.9.1.2 Request

Table 17-19: eventNotification() Request

Parameter	Required (M/O/C)	Type	Description	Domain Version
keyID	M	string	The unique identifier for a key. Computed as the 160-bit SHA-1 hash of the value of the bit string subjectPublicKey from the keyData (excluding the tag, length, and number of unused bits).	V-OD-FW
status	M	string	See Table 17-56	DS-VS
uiBundle	O	Object: UiBundle	See Table 17-46	DS-VS
reasonType	M	string	See Table 17-60	DS-VS
reasonSubtype	C	string	See Table 17-61	DS-VS
encryptedDevice Data	O	Object: EncryptedDataContainer	Contents of unencryptedDeviceData (Table 17-39) within EncryptedDataContainer (Table 17-37)	D-VS
encryptedConfig urationData	C	Object: EncryptedDataContainer	Request object for handle event notifications. SharingData object will only be present if receiving keyID is able to share. Content of unencryptedConfigurationData (Table 17-38) within	D-VS

Parameter	Required (M/O/C)	Type	Description	Domain Version
			EncryptedDataContainer (Table 17-37). Present if eventNotification ReasonType = KEY_CONFIGURATION_UPDATED	

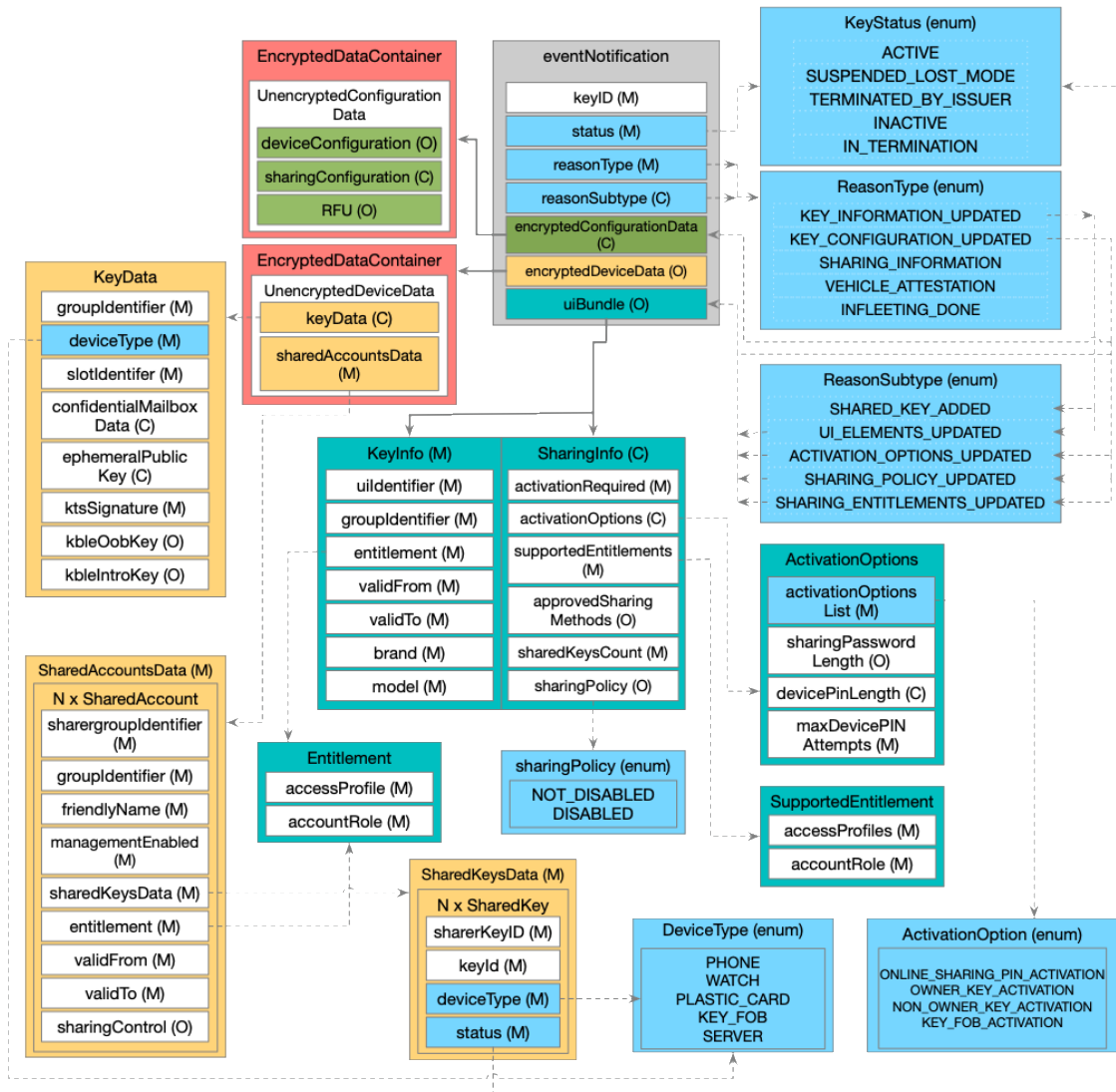
1 17.9.1.3 Response

2 Table 17-20: eventNotification() Response: Status Code 200: OK

Parameter	Required (M/O/C)	Type	Description	Domain Version
responseHeader	M	Object: ResponseHeader	ResponseHeader with status Code = 200: OK See Table 17-36	N/A

3

Figure 17-1: eventNotification Payloads



17.9.1.4 Samples

Listing 17-13: Sample eventNotification() Request

```

1. {
2.   "keyID": "6743E476477D34E52513353C0B81F0ABB2F1E33A",
3.   "status": "ACTIVE",
4.   "uiBundle": {
5.     "keyInfo": {
6.       "groupIdentifier": "3f66",
7.       "entitlement": {
8.         "accessProfile": 0,
9.         "accountRole": "88CF"
10.      },

```

```
11.     "validFrom": "2024-01-17T16:22:05.677Z",
12.     "validTo": "2024-01-17T16:22:05.677Z"
13. },
14.     "vehicleInfo": {
15.         "uiIdentifier": "8dcd8010-d3ac-4f24-bd94-de144e93acaf",
16.         "brand": "xyz",
17.         "model": "abc"
18.     },
19.     "sharingInfo": {
20.         "activationRequired": true,
21.         "supportedEntitlements": [
22.             "entitlement": {
23.                 "accessProfile": 0,
24.                 "accountRole": "88CF"
25.             }
26.         ],
27.         "shareableKeysCount": 3,
28.         "sharedKeysCount": 1
29.     }
30. },
31.     "reasonType": "KEY_INFORMATION_UPDATED",
32.     "reasonSubtype": "UI_ELEMENTS_UPDATED",
33.     "encryptedDeviceData": {
34.         "version": "ECIES_v1",
35.         "ephemeralPublicKey":
36.         "04613197827d91806d630bc4adff44686b012316eb03825f2d6587ffd58d32f4522ada80cc93679e
37.         1a316dc0729ebf8172fd41f0c0c1bdda01126f1a6186b2a008",
38.         "publicKeyHash":
39.         "eaf91Bd7947B5d25DF41f63ecc5915C47cB7BFDcaf939EfCeFc7cfA0bf7",
37.         "data": "kE8/Rs9sNo4olc6s3Qej7mXhAtDPcp ...1Bm/r7bKNX6m365vnIDEDc"
38.     }
39. }
```

1

2 17.10 APIs implemented by Server Based Devices

3 17.10.1 API – *inFleetExternal*

4 This API is offered by the SBOD using which the FMS can initiate the creation of an SBOD key.

5 17.10.1.1 Endpoint

6 HTTPS POST `/[service]/[version]/inFleetExternal()`

17.10.1.2 Request()

Table 17-21: *infleetExternal()* Request Body

Parameter	Required (M/O/C)	Type	Length	Description	Domain Version
vehicleId	M	string	17	The unique identifier of the vehicle. This should be the VIN of the vehicle, but any unique identifier may also be used.	
vehicleOEMIdentifier	C	string	4	Vehicle OEM Identifier as listed in [35] Included if vehicleId is not the VIN of the vehicle.	
proofOfOwnership	O	string	variable	Proof that the FMS owns the vehicle	

17.10.1.3 Response()

Table 17-22: *infleetExternal()* Response body: Status 200: OK

Parameter	Required (M/O/C)	Type	Description	Domain Version
responseHeader	M	Object: ResponseHeader	ResponseHeader with status Code = 200: OK See Table 17-36	N/A

17.10.2 API – prepareKeySharingExternal

This is a generic API offered by SBOD and SBFD for FMS and Service Provider to request the DK Sharing URL to be provided to the device of the vehicle user. This API is used during the DK sharing process as described in section 12.

17.10.2.1 End Point

HTTP POST `/[{service}]/[{version}]/prepareKeySharingExternal`

1 17.10.2.2 Request()

2 Table 17-23: prepareKeySharingExternal() Request body

Parameter	Required (M/O/C)	Type	Description	Domain Version
vehicleId	M	String	The unique identifier of the vehicle. This should be the VIN of the vehicle, but any unique identifier may also be used.	
accountInfoHash	C	String	Account Info Hash as defined in Section 6.3.4.3 . Included if needed by FMS.	
sharingId	M	String	The unique identifier of the digital key sharing assigned by FMS or Service Provider to create a mapping between the sharing session and the keyID.	
entitlements	M	Object: supportedEntitlements	Describes entitlement supported for the key being requested. See key configuration data field in ASN.1 as described in [41]	
activationOptionPolicy	O	enum	Present if second factor activation is required by FMS. See Table 17-66	

3 17.10.2.3 Response()

4 Table 17-24: prepareKeySharingExternal() Response Body: Status 200: OK

Parameter	Required (M/O/C)	Type	Description	Domain Version
responseHeader	M	Object: Response Header	ResponseHeader with status Code = 200: OK. See Table 17-36	N/A
sharingUrl	M	string	Cross platform sharing invitation URL as defined in chapter 11.2.1.1 to be forwarded to the fleet customer	
activationOptions	M	list	List of activation options as listed in Section 11.2 . See Table 17-62	
onlineSharingPIN	C	string	The Online Sharing PIN to be entered by the receiver on their device during sharing process. Required if activationOption = onlineSharingPinActivation (Section 11.5)	

5 17.10.3 API – manageKeyExternal

6 This is a generic API offered by SBOD and SBFD server for FMS and Service Provider to
 7 manage the lifecycle of their shared DKs. This API is used during a termination process as
 8 described in section 4.4.3.

17.10.3.1 EndPoint

HTTPS POST /{service}/{version}/manageKeyExternal

17.10.3.2 Request()

Table 17-25: manageKeyExternal() Request Body

Parameter	Required (M/O/C)	Type	Description	Domain Version
vehicleId	C	String	The unique identifier of the vehicle. This should be the VIN of the vehicle, but any unique identifier may also be used. Exactly one of vehicleId, keyID or sharingId shall be provided	
keyID	C	String	KeyId of the shared key. Exactly one of vehicleId, keyID or sharingId shall be provided	
sharingId	C	String	The unique identifier of the shared DK assigned by FMS to create a mapping between the sharing session and the keyID. Exactly one of vehicleId, keyID or sharingId shall be provided	
action	M	enum	Action of manage key (See Table 17-59).	
sbxdRemoteTerminationRequest	M	Object	See Table 17-52	

17.10.3.3 Response ()

Table 17-26: manageKeyExternal() Response Body: Status 200: OK

Parameter	Required (M/O/C)	Type	Description	Domain Version
responseHeader	M	Object: Response Header	ResponseHeader with status Code = 200: OK See Table 17-36	N/A
status	O	string	See Table 17-56	
encryptedDevice Data	M	Object: EncryptedDataContainer	Contents of unencryptedDeviceData (Table 17-39) within EncryptedDataContainer (Table 17-37)	D-VS

17.10.4 API – eventNotificationExternal

This is a generic API offered by FMS or Service Provider for SBOD, SBFD to forward notifications about different events during vehicle & Digital Key lifecycle.

17.10.4.1 End Point

HTTP POST /{service}/{version}/eventNotificationExternal

17.10.4.2 Request ()

Table 17-27: eventNotificationExternal() Request Body

Parameter	Required (M/O/C)	Type	Description	Domain Version
vehicleId	M	String	The unique identifier of the vehicle. This should be the VIN of the vehicle, but any unique identifier may also be used.	
uiBundle	O	Object: UiBundle	See Table 17-46	DS-VS
reasonType	M	string	See Table 17-60	DS-VS
encryptedDeviceData	O	Object: Unencrypted DeviceData	See Table 17-39	

17.10.4.3 Response ()

Table 17-28: eventNotificationExternal() Response body: Status 200: OK

Parameter	Required (M/O/C)	Type	Description	Domain Version
responseHeader	M	Object: ResponseHeader	ResponseHeader with status Code = 200: OK See Table 17-36	N/A

17.10.5 API – getKeyInformation

This is a generic API offered by SBOD to FMS to request information about all DKs linked to a vehicle.

17.10.5.1 End Point

HTTP POST /{service}/{version}/getKeyInformation

17.10.5.2 Request ()

Table 17-29: getKeyInformation() Request Body

Parameter	Required (M/O/C)	Type	Description	Domain Version
vehicleId	M	String	The unique identifier of the vehicle. This should be the VIN of the vehicle, but any unique identifier may also be used.	

17.10.5.3 Response ()

Table 17-30: getKeyInformation() Response Body: Status 200: OK

Parameter	Required (M/O/C)	Type	Description	Domain Version
responseHeader	M	Object: Response Header	ResponseHeader with status Code = 200: OK. See Table 17-36	N/A
SharedAccountsData	M	List	See Table 17-41	

17.10.6 API - defleetExternal

This is a generic API offered by vehicle OEM server for FMS to request vehicle unpairing with FMS. This API is used during the vehicle defleeting process as described in section 4.5.5.

17.10.6.1 End Point

HTTPS POST /{service}/{version}/defleet

17.10.6.2 Request ()

Table 17-31: deFleet() Request Body

Parameter	Required (M/O/C)	Type	Description	Domain Version
vehicleId	M	String	The unique identifier of the vehicle. This should be the VIN of the vehicle, but any unique identifier may also be used.	

17.10.6.3 Response ()

Table 17-32: deFleet() Response Body: Status 200: Ok

Parameter	Required (M/O/C)	Type	Description	Domain Version
responseHeader	M	Object: ResponseHeader	ResponseHeader with status Code = 200: OK See Table 17-36	N/A

17.10.7 API – cancelService

This is a generic API offered by SBFD for Service Provider to request service cancellation. This API is used during the service cancellation process as described in section [12.4.5](#).

17.10.7.1 End Point

HTTPS POST /{service}/{version}/cancelService

17.10.7.2 Request ()

Table 17-33: cancelService() Request Body

Parameter	Required (M/O/C)	Type	Description	Domain Version
vehicleId	M	string	The unique identifier of the vehicle. This should be the VIN of the vehicle, but any unique identifier may also be used.	

17.10.7.3 Response ()

Table 17-34: cancelService() Response Body: Status 200: OK

Parameter	Required (M/O/C)	Type	Description	Domain Version
responseHeader	M	Object: ResponseHeader	ResponseHeader with status Code = 200: OK See Table 17-36	N/A

17.11 Data Structures

17.11.1 Objects

17.11.1.1 RequestHeader

All requests from Server entities described in this section shall contain HTTP re-quest headers as defined below.

Table 17-35: HTTP Request Header

Parameter	Description	Required (M/O/C)
x-requestId	All requests by Device OEM Servers and Vehicle OEM Servers shall have an HTTP header “x-requestId”. The value shall be a UUID of length 36 containing hyphens.	M
x-timestamp	All requests by Device OEM Servers and Vehicle OEM Servers shall have an HTTP header “x-timestamp”. Timestamp should be specified in Unix timestamp. The value shall be given in milliseconds. The value shall not be	M

Parameter	Description	Required (M/O/C)
	modified in case of request retry. This is used to identify outdated requests in case of retry.	
x-device-oemId	All requests by Device OEM Servers shall have an HTTP header “x-device-oemId”. Value is the name of the Device OEM.	C
x-fmsId	All requests from FMS Servers shall have an HTTP header “x-fmsId”. Value is the name of the FMS.	C
x-sbfdId	All requests from SBFD shall have an HTTP header “x-sbfdId”. Value is the name of the SBFD.	C
x-sbodId	All requests from SBOD shall have an HTTP header “x-sbodId”. Value is the name of the SBOD.	C
x-spsId	All requests from Service Provider Servers shall have an HTTP header “x-spsId”. Value is the name of the Service Provider Server.	C
x-vehicle-oemId	All requests by Vehicle OEM Servers shall have an HTTP header “x-vehicle-oemId”. Value is the Vehicle OEM identifier (See Table 2-1 of [35]).	C
x-user-identifier	Device OEM server may include a HTTP header “x-user-identifier”. Value is - generated by the device OEM server and stored by Vehicle OEM servers - a AES-GCM encrypted base64 encoded string representation of a hash value generated using a pseudonymized device side userId. - up to 128 bytes long If included by Device OEM server, then all server API messages transmitted to device OEM server by Vehicle OEM server shall include the value.	O
x-device-oem-host	The FQDN of the Device OEM’s endpoint. Vehicle OEM uses this information to initiate communication to Device OEM for a specific keyID. The “x-device-oem-host” shall be sent by the Device OEM Server	M

1

2 17.11.1.2 ResponseHeader

3 All responses from the Vehicle OEM Server and Device OEM Server shall contain header data
4 of type “ResponseHeader” within the payload.

5

Table 17-36: ResponseHeader

Parameter	Required (M/O/C)	Type	Description	Domain Version
x-responseId	M	string	The corresponding response to a API request call shall have HTTP header “x-responseId”, which should echo the value of “requestId” in the request header. This is used to identify the request associated to the response for a particular API request and response pair. The x-responseId shall be sent by Device OEM Server and Vehicle OEM Server.	
statusCode	M	int	See Table 17-67	DS-VS

Parameter	Required (M/O/C)	Type	Description	Domain Version
subStatusCode	C	int	Included when statusCode is not equal to 200: OK. See Table 17-68	DS-VS
statusMessage	O	string	Maximum length = 256. Not parsed programmatically. Example: “Downstream system offline”	DS-VS

1

2 17.11.1.3 EncryptedDataContainer

3 This object is used by APIs to deliver encrypted data. The parameter “data” in [Table 17-37](#)
4 contains encrypted versions of API specific unencrypted objects.

5

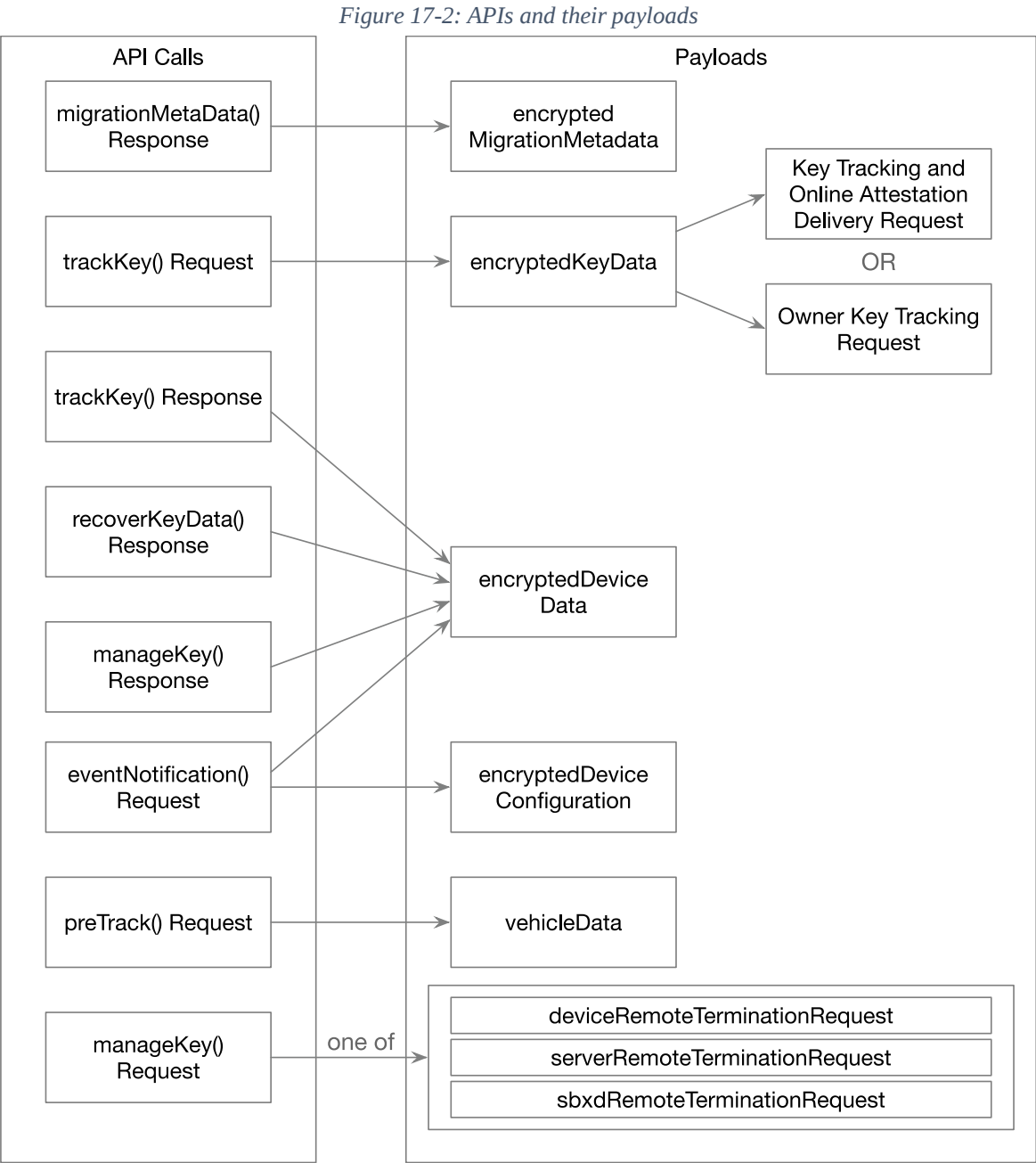
Table 17-37: EncryptedDataContainer

Parameter	Required (M/O/C)	Type	Description	Domain Version
version	M	string	Identifier for the encryption algorithm version. Version = ECIES_v1	
ephemeralPublicKey	M	string	Hex-encoded sender's ephemeral public key. This should be from an ephemeral key pair generated for a single message. Format for encoding the sender public key: concat(0x04, x, y) (This will be 65 bytes total for a key generated based on named curve secp256r1 - 1.2.840.10045.3.1.7). See Device.Enc.PK as defined in Section 14 .	
publicKeyHash	M	string	Hex-encoded recipients' key agreement public key fingerprint. Fingerprint generation algorithm: sha256Hash(toUncompressedRawECPublicKey-Format(recipientKAPublicKey)). SHA-256 hash of the uncompressed key agreement public key (concat(0x04, x, y)) of the receiving system used in ECIES_v1 Key Agreement.	

Parameter	Required (M/O/C)	Type	Description	Domain Version
data	M	Byte[]	Base64-encoded encrypted data. data=base64Encode(encrypt(unencryptedData, symmetricEncryptionParams)). UnencryptedData = oneof(UnencryptedDeviceData, UnencryptedDeviceConfigurationData (Table 17-38), UnencryptedMigrationMetadata, Owner keyTrackingRequest (Section 6.3.4.3) including outer tag 7F3E _h , keyTrackingResponse (See Table 11-19 and Table 17-40) Unencrypted signedSharingSecInfo (Table 11-23) including outer tag 7F41 _h) See Figure 17-2	

1

1



2

3 17.11.1.4 UnencryptedConfigurationData

4

5

Table 17-38 UnencryptedDeviceConfigurationData

Parameter	Required (M/O/C)	Type	Description	Domain Version
deviceConfiguration	O	String	TLV of device configuration data including contents of Table 5-14 with the outer tag 7F4E _h . This field shall not be used to modify device wireless capabilities. If modified device wireless	

Parameter	Required (M/O/C)	Type	Description	Domain Version
			capabilities are included, then the recipient shall ignore the modifications.	
sharingConfiguration	C	String	TLV of sharing configuration including contents of Tag 7F60 _h in Table 5-14 including outer tag 7F60 _h . Present, if deviceConfiguration (Tag 7F4E _h) is not present. Absent, if deviceConfiguration is present.	
RFU	O	string	Optional values for future use	

Device shall overwrite all the current values of Tag 7F60_h and Tag 7F4E_h stored in the device with the new values received in the UnencryptedDeviceConfigurationData (see [Table 17-38](#)) within EncryptedDataContainer, if eventNotification reasonType = KEY_CONFIGURATION_UPDATED.

17.11.1.5 UnencryptedDeviceData

Object that contains content of trackKey Response encrypted to the device.

Table 17-39: UnencryptedDeviceData

Parameter	Required (M/O/C)	Type	Description	Domain Version
keyData	C	object	Object of type KeyData. See Table 17-40 . KeyData includes all elements necessary to enable the tracked key on the mobile device. Mandatory when keyData is transmitted for private vehicles.	
sharedAccountsData	M	object	Object of type SharedAccountsData. This object shall only be sent to keys that can share keys to other accounts and/or have visibility of other accounts. If no shared accounts exist, that can be managed or viewed by this account, then sending entity includes an empty list. See Table 17-41	
intraAccountKeysData	M	object	Object of type SharedKeysData(Table 17-43). Contains the list of shared keys for this account, except the current key.	

17.11.1.6 KeyData

This data structure matches Owner Key Tracking Response parameters outlined in [Table 6-3](#) or Key Tracking Response parameters outlined in [Table 11-11](#)

Table 17-40: keyData

Parameter	Required (M/O/C)	Type	Description	Format
groupIdentifier	M	string:	GroupIdentifier of the key referenced by the keyID included in the API response data. Value	byte

Parameter	Required (M/O/C)	Type	Description	Format
		2 bytes	generated and assigned by the Vehicle OEM. See Section 6.3.4.3 . This field is a unique identifier over (vehicle, accountInfoHash).	
deviceType	M	string	See Table 17-57	
slotIdentifier	M	string Maxlen:8	The slot identifier to assign for the key (see Table 6-3 and Table 11-11).	byte
confidentialMailboxData	C	string	Encrypted confidential mailbox data (see Table 6-3 and Table 11-19). Present if immobilizer tokens are retrieved online.	byte
ephemeralPublicKey	C	String	Hex-encoded sender's ephemeral public key. This should be from an ephemeral key pair generated for a single message. Format for encoding the sender public key: concat(0x04, x, y) (This will be 65 bytes total for a key generated based on named curve secp256r1 - 1.2.840.10045.3.1.7). Present if immobilizer tokens are retrieved online.	byte
ktsSignature	M	String Maxlen: 320 bytes	Signature created by key tracking server (KTS); see Section 6.3.4.4 .	byte
kBleOobKey	O	String	Online Kble_oob key for BLE connection to vehicle; see Section 19.5 .	byte
kBleIntroKey	O	string	Online Kble_intro key for BLE connection to vehicle; see Section 19.5 .	byte

1

2 17.11.1.7 SharedAccountsData

3

4

Table 17-41:SharedAccountsData

Parameter	Required (M/O/C)	Type	Description	Domain Version
sharedAccountsData	M	List of objects	List of objects of type SharedAccount. See Table 17-42 . A list containing information about all associated shared accounts. Each element in the list represents a shared account and includes keys based on the visibility capability of the account. This object shall only be sent to devices that can share keys to other accounts and/or have visibility of other accounts.	

Parameter	Required (M/O/C)	Type	Description	Domain Version
			If no shared accounts that can be managed or viewed by this account exist, then the sending entity includes an empty list.	

1

2 17.11.1.8 SharedAccount

3

Table 17-42: SharedAccount

Parameter	Required (M/O/C)	Type	Description	Format
sharerGroupIdentifier	M	string: 2 bytes	GroupIdentifier of the account that shared to this account. Value generated and assigned by the Vehicle OEM. See Section 11.4.4 . This field is a unique identifier over (vehicle, accountInfoHash).	byte
groupIdentifier	M	string: 2 bytes	Group Identifier of the key referenced by the keyID included in the API response data. Value generated and assigned by the Vehicle OEM. See Section 11.4.4 . This field is a unique identifier over (vehicle, accountInfoHash).	Byte
friendlyName	M	String Maxlen: 30	The friendly receiver's name to be displayed to the user.	byte
managementEnabled	M	boolean	If true, recipient can manage keys of the account referenced by this SharedAccount instance in the API response data. See Section 11.8	
sharedKeysData	M	object	Object of type SharedKeysData(Table 17-43). Contains the list of shared keys for this account	byte
entitlement	M	object	Object of type Entitlement. See Table 17-49	byte
validFrom	M	string	Key validity start date. Time should be specified in UTC ISO-8601 Format yyyy-MM-dd'T'HH:mm:ss.SSSZ.	date-time
validTo	M	string	Key validity end date. Time should be specified in UTC ISO-8601 Format yyyy-MM-dd'T'HH:mm:ss.SSSZ.	date-time
sharingControl	O	Enum (max length 32 bytes)	`VEHICLE` - This Digital Key can enable and disable Digital Key sharing in the Vehicle UI. `VEHICLE_AND_DEVICE` - This Digital Key can enable and disable Digital Key sharing in the Vehicle UI_and_ Device UI. If this field is absent, then this Vehicle or the Digital Key's `accountRole` cannot control Digital Key sharing (see Section 12.4.6). (see section 12.4.6).	byte

17.11.1.9 SharedKeysData

List of all associated shared keys

Table 17-43: Shared Keys Data

Parameter	Required (M/O/C)	Type	Description	Domain Version
sharedKeysData	M	List of objects	List of objects of type SharedKey. See Table 17-44	

17.11.1.10 SharedKey

Object that describes a shared key.

Table 17-44: SharedKey

Parameter	Required (M/O/C)	Type	Description	Domain Version
sharerKeyID	M	string	The unique identifier for a key. Computed as the 160-bit SHA-1 hash of the value of the bit string subjectPublicKey from the keyData (excluding the tag, length, and number of unused bits).	V-OD-FW
keyID	M	string	The unique identifier for a key. Computed as the 160-bit SHA-1 hash of the value of the bit string subjectPublicKey from the keyData (excluding the tag, length, and number of unused bits).	V-OD-FW
status	M	object: KeyStatus	See Table 17-56	
deviceType	M	string	See Table 17-57	

17.11.1.11 UnencryptedMigrationMetadata

Object with contents of migrationMetadata() Response. This is delivered to the device within an encryptedDataContainer.

Table 17-45: UnencryptedMigrationMetadata

Parameter	Required (M/O/C)	Type	Description	Format
endpointCreationData	M	string	TLV of EndpointCreationData including outer tag 7F27 _h . Further info can be found in Table 15-13.	byte
deviceConfiguration	C	string	TLV of DeviceConfigurationData including outer tag 7F4E _h . Present if sharingConfiguration is absent. See Table 5-14	byte
mailboxMapping	M	string	TLV of MailboxMapping including outer tag 7F4D _h . See Table 5-13 .	byte

Parameter	Required (M/O/C)	Type	Description	Format
sharingConfiguration	C	string	See description in Table 17-38 . Also see Tag 7F60 _h (Table 5-14)	byte

17.11.1.12 UiBundle

Table 17-46: UiBundle

Parameter	Required (M/O/C)	Type	Description	Domain Version
vehicleInfo	M	object	Object of type VehicleInfo. See Table 17-55	
keyInfo	M	Object	Object of type KeyInfo. See Table 17-47	
sharingInfo	C	Object	Object that contains all UiBundle data related to sharing of the key. Response object when the signing key of the SBOD was successfully sent to the vehicle. Included only if the key can be shared to additional recipients. See Table 17-48	

17.11.1.13 KeyInfo

Table 17-47: KeyInfo

Parameter	Required (M/O/C)	Type	Description	Format
groupIdentifier	M	String: 2 bytes	Group Identifier of the key referenced by the keyID included in the API response data. Value generated and assigned by the Vehicle OEM. See Section 11.4.4 . This field is a unique identifier over (vehicle, accountInfoHash).	byte
entitlement	M	Object: Entitlement	See Table 17-49	
validFrom	M	string	Key validity start date. Time should be specified in UTC ISO-8601 Format yyyy-MM-ddT'HH:mm:ss.SSSZ.	date-time
validTo	M	string	Key validity end date. Time should be specified in UTC ISO-8601 Format yyyy-MM-ddT'HH:mm:ss.SSSZ.	date-time

17.11.1.14 SharingInfo

Table 17-48: SharingInfo

Parameter	Required (M/O/C)	Type	Description	Format
activationRequired	M	Boolean	Indicates that a 2nd factor is required to activate a shared key.	
activationOptions	C	Object: Activation Options	Provides the list of all supported 2nd factor activation options for activation of a shared key. Present if activationRequired is true. See Table 17-51	
supportedEntitlements	M	List of Objects	Contains all possible objects of type SupportedEntitlement, that a key shared by this account can have. See Table 17-50	
approvedSharingMethods	O	List of Strings	Describes the list of all approved sharing method group identifiers as described in Table 11-11 (list of string encoded hex values of tags 61 _h , without tag and length of the subtags). Example: “ 4002000041020001”	
shareableKeysCount	M	integer	Total number of remaining keys that can be shared. Key sharing should only be offered to the user if value is greater than 0.	
sharedKeysCount	M	integer	Amount of keys that have been shared.	
sharingPolicy	O	Enum	See Table 17-63 The absence of this field means that Digital Key sharing is NOT DISABLED (i.e., enabled).	byte

17.11.1.15 Entitlement

Table 17-49: Entitlement

Parameter	Required (M/O/C)	Type	Description	Format
accessProfile	M	integer	Single Standard access profile as outlined in Table 11-21	
accountRole	M	String	The role of the account which applies to all devices of this account. See Section 2.8.4 .	

17.11.1.16 SupportedEntitlement

Table 17-50: SupportedEntitlement

Parameter	Required (M/O/C)	Type	Description	Format
accessProfiles	M	List of integers	List of all access profiles supported by vehicle that can be used for this accountRole. See Table 11-21.	
accountRole	M	String	The accountRole which can be shared with another account. See Section 2.8.4 .	

17.11.1.17 ActivationOptions

Table 17-51: ActivationOptions

Parameter	Required (M/O/C)	Type	Description	Format
activationOptionsList	M	List of Objects	Contains a list of Objects of type ActivationOption that may be supported by the vehicle. Provides the list of all supported 2nd factor activation options for activation of a shared key (17.8.2.1). See Table 17-62	
sharingPasswordLength	C	integer	Required if activationOptionsList contains ONLINE_SHARING_PIN_ACTIVATION. See Section 11.2.1.6	
maximumOnlineSharingPinAttempts	C	integer	Gives the maximum number of retries allowed for Online Sharing PIN input by receiver of a shared key. Present if activationOption in preTrack() request (See 17.7.6.4) contains ONLINE_SHARING_PIN_ACTIVATION. Value of attemptsLeft is ≥ 0 and < 10 .	

17.11.1.18 sbxdRemoteTerminationRequest

Provided by SBOD or SBFD, this object contains the Remote Termination Request for termination of a remote shared key. This is the JSON formatted version of deviceRemoteTerminationRequest.

Table 17-52: SbxdRemoteTerminationRequest

Parameter	Required (M/O/C)	Type	Description	Format
keyPairs	C	List of Objects	List of all keys to be deleted. See Table 17-54 Either keyPairs or GroupTerminations shall be present	
GroupTerminations	C	List of Objects	List of all groups to be deleted. See Table 17-53 Either keyPairs or GroupTerminations shall be present	

17.11.1.19 GroupTermination

Table 17-53: GroupTermination

Parameter	Required (M/O/C)	Type	Description	Format
GroupIdentifier	M	string	Group Identifier of the key referenced by the keyID included in the API response data. Value generated and assigned by the Vehicle OEM. See Section 11.4.4. This field is a unique identifier over (vehicle, accountInfoHash).	
DeleteSubtree	M	Boolean	Indicates that the entire subtree of the account shall be deleted as well	

17.11.1.20 KeyPair

Table 17-54: KeyPair

Parameter	Required (M/O/C)	Type	Description	Format
keyID	M	string	As defined in Table 17-1	
slotIdentifier	M	string	The slot identifier to assign for the key (see Table 6-3 and Table 11-19).	

17.11.1.21 vehicleInfo

Table 17-55: VehicleInfo

Parameter	Required (M/O/C)	Type	Description	Format
uiIdentifier	M	string MaxLen: 32	Identifier referencing visual elements that are required to create user interface for Digital Key. The definition and encoding of the string value is agreed between vehicle and Device OEMs individually and is out of scope of this specification.	
brand	M	string maxlen: 32	The brand of the vehicle	
model	M	String maxlen: 32	The model of the vehicle	

17.11.2 Enums

17.11.2.1 KeyStatus

A Digital Key can have the states in the following table. Throughout the lifetime of a Digital Car Key its state will change according to the following:

- After creation the key is in the state INACTIVE. If no further activation is necessary, the key is in ACTIVE state right after creation.
- Subject to Vehicle OEM policy, additional steps may be required to activate the key. These activation options are provided to the device using the object SharingInfo ([Table 17-48](#)).
- After activation (vehicle OEM policy applies) the key is in the status ACTIVE.
- If the key is set to lost mode by the user, its state changes to SUSPENDED_LOST_MODE. (Attention: This does not need to be supported by all vehicle OEMs.)
- If the key is deleted by a party that is not able to delete the key right away (fade out period applies), it changes its status into IN_TERMINATION.
- As soon as the key is deleted by either the keyholding device itself or by the vehicle, so that the key is not usable anymore, the state of the key changes into its final state deleted (TERMINATED_BY_ISSUER).

The list of SharedKeys (e.g., returned in encryptedDeviceData (see [Table 17-19](#) shall not contain any key in “Terminated” state.

Table 17-56: KeyStatus

Parameter	Type	Permitted Values		Domain Version
		Status	Status ID	
KeyStatus	enum	ACTIVE	1	DS-VS
		SUSPENDED_LOST_MODE	7	
		TERMINATED_BY_ISSUER	10	
		INACTIVE	14	
		IN_TERMINATION	15	

17.11.2.2 DeviceType

Table 17-57: DeviceType

Parameter	Type	Permitted Values	Domain Version
DeviceType	enum	PHONE WATCH PLASTIC_CARD KEY_FOB SERVER	DS-VS

17.11.2.3 *KeyOrigin*

Table 17-58: *KeyOrigin*

Parameter	Type	Permitted Values	Domain Version
keyOrigin	enum	OWNER_PAIRING SHARED MIGRATION	DS-VS

17.11.2.4 *manageKey Actions*

SUSPEND and RESUME actions are only applicable in manageKey API calls from Device OEM server to Vehicle OEM Server.

Table 17-59: *Action*

Parameter	Type	Permitted Values	Domain Version
Action	enum	TERMINATE SUSPEND RESUME	DS-VS

17.11.2.5 *eventNotification Reasons*

Reasons for an eventNotification call from Vehicle OEM Server to Device OEM Server are listed in [Table 17-60](#).

Table 17-60: *ReasonType enum*

Parameter	Type	Permitted Values	Domain Version
ReasonType	enum	KEY_INFORMATION_UPDATED KEY_CONFIGURATION_UPDATED SHARING_INFORMATION VEHICLE_ATTESTATION INFLEETING_DONE	V-OD-FW

Table 17-61: *ReasonSubtype associated with ReasonType*

Parameter	Type	Permitted Values	Associated ReasonType	Domain Version
ReasonSubtype	enum	SHARED_KEY_ADDED	KEY_INFORMATION_UPDATED	V-OD-FW
		UI_ELEMENTS_UPDATED		
		ACTIVATION_OPTIONS_UPDATED	SHARING_INFORMATION	
		SHARING_POLICY_UPDATED		
		SHARING_ENTITLEMENTS_UPDATED		

17.11.2.6 *activationOption*

Table 17-62: *activationOption*

Parameter	Type	Permitted Values	Domain Version
activationOption	Enum	ONLINE_SHARING_PIN_ACTIVATION OWNER_KEY_ACTIVATION RECEIVER_KEY_ACTIVATION KEY_FOB_ACTIVATION SHARING_PASSWORD_ACTIVATION ⁷	V-OD-FW

17.11.2.7 *sharingPolicy*

Table 17-63: *sharingPolicy*

Parameter	Type	Permitted Values	Domain Version
sharingPolicy	enum	ENABLE DISABLED_SERVER_ONLY DISABLED	V-OD-FW

⁵ `ENABLED` means the Digital Key sharing is enabled right now. The vehicle accepts shared
⁶ Digital Keys right now.

⁷ `DISABLED\_SERVER\_ONLY` means the Digital Key sharing is disabled right now and
⁸ trackKey will return an error. Since the vehicle was not yet reached, it might still accept already
⁹ shared Digital Keys right now.

¹⁰ `DISABLED` means the Digital Key sharing is disabled right now. The vehicle does not accept
¹¹ shared Digital Keys right now.

¹² The absence of this field means that Digital Key sharing is `ENABLED` (i.e., enabled).

17.11.2.8 *preTrackDataType*

Table 17-64: *preTrackDataType*

Parameter	Type	Permitted Values	Domain Version
preTrackDataType	enum	UI_BUNDLE ONLINE_SHARING_PIN	V-OD-FW

⁷ Since LONG_TERM_SHARED_SECRET is only available on the device that was used for owner pairing, SHARING_PASSWORD_ACTIVATION shall be offered by Vehicle OEM to the original owner paired device only.

17.11.2.9 *preShareDataType*

Table 17-65: *preShareDataType*

Parameter	Type	Permitted Values	Domain Version
preShareDataType	enum	SENDER_ONLINE_SHARING_PIN RECEIVER_PUBLIC_KEY RECEIVER_ACCOUNT_INFO_HASH	DS-VS

17.11.2.10 *activationOptionPolicy*

Table 17-66: *activationOptionPolicy*

Parameter	Type	Permitted Values	Domain Version
activationOptionPolicy	enum	MFA_NOT_ENFORCED MFA_ENFORCED	V-OD-FW

17.11.3 Server Status Codes

The default “Caller should retry” policy of a given status code is shown in this section, which shall be followed unless otherwise specified with a sub-status code described in Table 17-68

When multiple HTTP Status Codes are permitted, caller (e.g., Vehicle OEM) server should choose the HTTP StatusCode that fits the specific error (5xx for errors within Vehicle OEM server, 4xx for errors on caller’s side, e. g., Device OEM server).

Status code 50000 Unexpected error – shall not be used unless there are no more appropriate sub-status codes defined. Callers shall accept unknown sub-status codes and map them to “unexpected error” internally to avoid the use of versioning to introduce new error codes.

The caller shall retry any request satisfying one or more of the following criteria:

- Received an HTTP status code that allows a retry (see [Table 17-68](#))
- Request timed out or otherwise failed at the network level such that no HTTP response was received or processed.

Requests may be retried a maximum of three times. Retry attempts shall adhere to the following guidelines:

- First retry shall occur a pseudorandom amount of time Δt after the original attempt, where Δt is between 0s–1s inclusive.
- Second retry shall occur a pseudorandom amount of time Δt after the first retry attempt, where Δt is between 1s–5s inclusive.

Third retry shall occur a pseudorandom amount of time Δt after the second retry attempt, where Δt is between 5s–30s inclusive.

1

Table 17-67: Status Code Values

Value	Description	Called Should Retry
200	Request was received and processed	No
301	A resource is moved permanently, retry on new location	Yes
302	A resource is moved temporarily, retry on new location	Yes
400	Bad Request	No
404	Malformed or unknown URI	No
500	Internal server error	Yes
503	Service unavailable	Yes

2

3

Table 17-68: Server SubStatus Codes

Sub Status Code	HTTP Status Code	Caller Should Retry	Description	Usage
50000	400 / 500	Yes	Unexpected error	Shall not be used unless a more appropriate sub-status code has not been defined
50001	500	Yes	Downstream systems unavailable	Shall be used when systems on callee side are temporarily unavailable for unknown reasons and a later retry will likely succeed. There will be no immediate retries
50002	503	No	OEM scheduled downtime	Shall be used when systems on callee side are temporarily unavailable for unknown reasons and a later retry will likely succeed. There will be no immediate retries
50009	404	Yes	No associated user	Shall be used when there is no user account registered
50110	404	No	Unknown key id supplied	Shall be used when the referred key id is not known (or has already been deleted) on callee side. If possible, a more detailed sub-status code shall be used to refer to either owner or receiver key id
50111	404	No	Unknown owner Digital Key identifier supplied	Shall be used when the owner key id is not known (or has already been deleted) on callee side
50112	404	No	Unknown Digital Key identifier of a receiver supplied	Shall be used when the receiver's key id is not known (or has already been deleted) on callee side

Sub Status Code	HTTP Status Code	Caller Should Retry	Description	Usage
50113	400	No	Maximum sharing key limit reached	Shall be used when the received key tracking request refers to a key that would exceed the vehicle's capacity of storing keys (or hit a business limit). This error should be avoided by restricting the number of shared keys to the allowed limit on device side
50114	400 / 500	No	Cryptography error occurred	Shall be used when any cryptographic element (signature) in the supplied data is wrong
50115	500	No	Feature not supported	Shall be used when the caller data refers to a feature that is not supported on callee side
50116	400	N/A	Conflicting Key Slot Identifier	Shall be used when there is a mismatch between the key id and the expected slot identifier in the trackKey() request (e.g., slot identifier was used with another key id before)
50117	400 / 500	No	Version not supported	Shall be used when the version sent in the request is not supported by the recipient
50118	400 / 500	No	Invalid or expired Owner Instance CA Certificate	Shall be used when the request contains an invalid or expired Owner Instance CA Certificate
50119	400	No	Invalid or expired Sharee Instance CA Certificate	Shall be used when Instance CA Certificate from Sharee has expired or is invalid
50120	400	No	Invalid or expired Device OEM Root Certificate	Shall be used when root certificate of the Device OEM has expired or is invalid
50121	400	No	Invalid or expired Vehicle OEM RootCertificate	Shall be used when root certificate of the Vehicle OEM has expired or is invalid
50122	400	No	Owner Pairing not possible: vehicle is not ready	Shall be used in case the vehicle is unmapped from account after owner pairing but before key tracking
50123	400	No	Receiver tracking failed: Owner signature not as expected	Shall be used when owner signature is invalid
50124	400	No	Receiver tracking failed: No owner key found or in invalid state	Shall be used when owner key is in invalid state

Sub Status Code	HTTP Status Code	Caller Should Retry	Description	Usage
50125	301	Yes	Key references are not available on the given DS-VS version of the API	Shall be used when a key is accepting requests on a different DS-VS version of the API. Use `Location` header to indicate the correct API version
50126	400 / 500	No	Invalid Remote Termination Request	Shall be used when ManageKey Request contains a deviceRemoteTerminationRequest or serverRemoteTerminationRequest that is incorrectly formatted
50127	403	No	Insufficient Management Rights	Shall be used when device attempts to delete a key that it does not have management rights to delete
50128	400	No	Invalid Migration Version	Shall be used when device attempts to migrate incompatible key versions
50129	400	No	Migration not available	Shall be used when device attempts full migration before the owner device has performed full migration
50130	400	No	Migration failed on the vehicle server	Shall be used as response to migrationMetadata() request and trackKey() in migration, if migration has failed on the vehicle server
50131	400	No	Receiver tracking failed: Diamond sharing occurred	Shall be used when Vehicle Server detects a Digital Key for the Vehicle exists on the same or different Device within the same Device OEM account
50132	400	No	Digital Key Sharing currently disabled	Shall be used as response to preShare() and trackKey() if Digital Key sharing is currently disabled
50133	400	No	Shared tracking failed: No sharer key found or in invalid state	Shall be used when sharer key is in invalid state
50134	400	Yes	Wrong PIN entered	Shall be used when sharing recipient sends wrong PIN. Shall only be retried if `PreTrackResponse.attemptsLeft > 0`
50135	400	No	Conflicting roles between requested key and existing key	Shall be used when the roles conflict between the key requested in a trackKey() call and a key already existing on the caller's device
50136	422	Yes	Mandatory parameter is absent	Shall be used when mandatory parameter is absent in the request payload carried in an API call

1

2 *Note:* Please refer to Section [17.12](#) for encryption scheme and other details.

17.12 ECC Encryption for Device and Server

This section defines the specifics of the ECIES_v1 algorithm for ECDHE encryption and decryption.

17.12.1 Frame/Packet Contents

This following information shall be included in the frame/packet sent from the encrypting/sending party to the decrypting/receiving party:

1. Algorithm Identifier

The identifier for the algorithm specified is “ECIES_v1”.

2. Sender Key Agreement Public Key

The sender's key agreement public key shall be from an ephemeral key pair generated for a single message.

The format for encoding the sender public key: concat(0x04, x, y)

(This will be 65 bytes total for a key generated based on named curve secp256r1 and its OID value 1.2.840.10045.3.1.7.

3. Recipient Key Agreement Public Key Fingerprint

The recipient's key agreement public key fingerprint. The fingerprint shall be generated using algorithm: SHA-256 hash of the uncompressed key agreement public key (i.e., concat(0x04, x, y)) of the receiving party used in ECIES_v1 Key Agreement.

fingerprint = sha256Hash(toUncompressedRawECPublicKeyFormat(recipientKAPublicKey));

4. Encrypted Data

The binary encrypted data.

17.12.2 Encryption Process

The encrypting system shall follow the steps below to encrypt the data:

1. Generate sender's ephemeral EC keypair on named curve ephemeral for the message. secp256r1 and its OID 1.2.840.10045.3.1.7.

2. Using recipient's encryption public key and the ephemeral private key, generate the shared secret using Elliptic Curve Diffie-Hellman key agreement algorithm with OID 1.3.132.1.12.

3. Derive the symmetric key as described in Key Derivation Function (see Section [17.12.4](#))

4. Derive the initialization vector for symmetric encryption as described in initialization vector derivation (see Section [17.12.6](#)).

5. Encrypt data using all 128 bits of the derived symmetric key using AES-128 id-aes128-GCM with its OID 2.16.840.1.101.3.4.1.6 with no padding, the initialization vector, and no associated authentication data. The 16-byte GCM authentication tag shall be appended to the cipher text.

17.12.3 Decryption Process

Receiving system shall follow the steps below to decrypt the encrypted data:

11. Validate that the sender's ephemeral public key is valid on the named curve secp256r1 and its OID 1.2.840.10045.3.1.7.
12. Using receiving system's encryption private key (identified by recipient fingerprint) and the sender's ephemeral public key, generate the shared secret using Elliptic Curve Diffie-Hellman key agreement algorithm with OID 1.3.132.1.12.
13. Derive the symmetric key as described in Key Derivation Function (see Section 17.12.4).
14. Derive the initialization vector for symmetric encryption as described in Initialization vector derivation (See Section 17.12.6).
15. Decrypt the data using AES-128 id-aes128-GCM with its OID 2.16.840.1.101.3.4.1.6 with no padding, the initialization vector, and no associated authentication data. The 16-byte GCM authentication tag shall be appended to the cipher text.

17.12.4 Key Derivation Function

This subsection defines the algorithm for deriving the symmetric key from a shared secret and other parameters.

Derive the keying material according to the algorithm defined in Section 3.6 of [22], using the ANSI X9.63 Key Derivation Function described in Section 3.6.1 of [22].

```
Z = sharedSecretFromKeyAgreement(myECPrivateKey, otherPartyECPublicKey);
sharedInfo = computeSharedInfo();
counter = 0x00000001; // 4 bytes
keyingMaterial = Hash(Z | counter | sharedInfo);
```

where

senderKeyAgreementPrivateKey is generated in item 2 of Section 17.12.1

recipientKeyAgreementPublicKey is defined in item 3 of Section 17.12.1.

computeSharedInfo is defined in Section 17.12.4.1.

with the following parameters:

Input name	Value
H (Hash function)	SHA-256
keydatalen (Length of symmetric key)	16
Z (Shared secret)	The shared secret calculated using the ECDH key agreement keys
SharedInfo	The shared information containing the Shared Info sequence

17.12.4.1 Shared Info

Input name	Required	Value
Sender Key Agreement Public Key	Mandatory	The senders' ephemeral key agreement public key. Format: toUncompressedRawECPublicKeyFormat(sender.ephemeralKeyAgreement-PublicKey)

Recipient Key Agreement Public Key	Mandatory	The recipient's key agreement public key Format: toUncompressedRawECPublicKeyFormat(recipient.keyAgreementPublicKey)
------------------------------------	-----------	--

17.12.5 Symmetric Encryption Key

The first 16 bytes of the 32-byte keying material output from the KDF shall be used as the symmetric key for encryption.

17.12.6 Initialization Vector Derivation

The last 16 bytes of the 32-byte keying material output from the KDF shall be used as IV.

17.12.7 EC Public Key Point Encoding in Uncompressed Form

To encode the EC public key point in uncompressed form, follow Elliptic-Curve-Point-to-Octet-String Conversion in Section 2.3.3 of [\[22\]](#).

17.13 Example

This section shows the step-by-step output of the encryption of a sample message using sample sender and recipient keys.

17.13.1 Keys

Party	Key Type	Value
Sender	Key Agreement Key	
	Private Key Hex (S)	53994c02ca9fd1afbda1742f43d3c17dedfaf3367727208fe9cc50a33824a0
	Public Key Hex (0x04 x y)	0474542541492424edc34f33ba94e61bf718f33ca393d1bf0816f156e4a266873f59564aad1a95c9fd8971b527171784e390d9137d037fe2ae30490b1ed1d73aa3
Recipient	Key Agreement Key	
	Private Key (S)	6c300cac339b4c7084e34175e2e6f5e1d1b135d158bd578c2b1af870facaef17
	Public Key Hex (0x04 x y)	04ad2d126c3f8f85bf5796f6ab849b57a35133fed491eced0254ed6a1169d9281f98018f1aa71d222874d72f47b0b28d84dacb8801b96e815c06f1151210cc7090
	Public Key Fingerprint Hex	ac0095a2121357c69d64213137973222d9873bc80dbcc6ea044d6db3ec5bfbc

1 17.13.2 Message to encrypt

```
{ "id": "Hello", "value": "World" }
```

3 17.13.3 Encryption process

Data	Format	Value
Algorithm Identifier	String	ECIES_v1
Derived Shared Secret	Hex	a6c3021dc18ad03959768250e872818585264fbd1b6de6ed32a7eb16f19f858
KDF Shared Info	Hex	0474542541492424edc34f33ba94e61bf718f33ca393d1bf0816f156e4a26687 3f59564aad1a95c9fd8971b527171784e390d9137d037fe2ae30490b1ed1d73a a304ad2d126c3f8f85bf5796f6ab849b57a35133fed491eced0254ed6a1169d9 281f98018f1aa71d222874d72f47b0b28d84dacb8801b96e815c06f1151210cc 7090
Keying Material	Hex	58a5f8b53ff010a62eb2e98303f7d2d088b35fa8b758797777dbafa81df3bc9f
Derived AES Key	Hex	58a5f8b53ff010a62eb2e98303f7d2d0
Initialization Vector	Hex	88b35fa8b758797777dbafa81df3bc9f
Encrypted Data	Hex	46e9fbef564e5eba68c094da08172bac8299bd268749763224dea8a59313d548 963dafbc3dcc7ea206646edeb4469e5d6eed38

4 17.13.4 Decryption process

Data	Format	Value
Algorithm Identifier	String	ECIES_v1
Derived Shared Secret	Hex	a6c3021dc18ad03959768250e872818585264fbd1b6de6ed32a7eb16f19f858
KDF Shared Info	Hex	0474542541492424edc34f33ba94e61bf718f33ca393d1bf0816f156e4a266873f59564aad1a95c9fd8971b527171784e390d9137d037fe2ae30490b1ed1d73a a304ad2d126c3f8f85bf5796f6ab849b57a35133fed491eced0254ed6a1169d9 281f98018f1aa71d222874d72f47b0b28d84dacb8801b96e815c06f1151210cc 7090
Keying Material	Hex	58a5f8b53ff010a62eb2e98303f7d2d088b35fa8b758797777dbafa81df3bc9f
Derived AES Key	Hex	58a5f8b53ff010a62eb2e98303f7d2d0
Initialization Vector	Hex	88b35fa8b758797777dbafa81df3bc9f

Decrypted Data	String	{ "id": "Hello", "value": "World" }
----------------	--------	-------------------------------------

1
2

18 SECURITY

18.1 SPAKE2+ Protocol Description

18.1.1 General

This section explains the principles of the SPAKE2+ protocol. Refer to Section [18.4](#) for implementation details.

The system uses SPAKE2+, which is an ECC-based pairing algorithm protocol, to mutually authenticate two entities based on the knowledge of a password provided on the client side (i.e., device) and a verifier, permanently stored in the server (i.e., vehicle). For more detailed information, see [\[10\]](#).

The NIST P-256 curve shall be used [\[8\]](#).

The Vehicle OEM Server should generate the password and provision the necessary elements (w_0 , L ; see [18.1.2](#)) into the vehicle well before start of the owner pairing, so that pairing is possible even when the vehicle is offline at the moment of owner pairing.

The Script key derivation is executed in the server and on the device, which allows the server and device to adapt the derivation $\langle \rangle$ parameters over time to counter increases in attacker performance.

The password pwd should be provided to the owner through the Vehicle OEM account, protected by the login credentials known only by the owner. The password is UTF-8 encoded and should be passed into the script function in this coding.

All values are assumed to be in big-endian byte order. The x and y random generator shall have uniform distribution over the required range and be cryptographically secure.

18.1.2 Execution

The Vehicle OEM Server derives L and w_0 from the password as follows:

z_0 = left 40 bytes of $\text{Script}(pwd, s, N_{\text{script}}, r, p, dkLen)$

z_1 = right 40 bytes of $\text{Script}(pwd, s, N_{\text{script}}, r, p, dkLen)$

It shall derive w_0 and w_1 using the method FIPS 186-4, B.5.1 Per-Message Secret Number Generation Using Extra Random Bits:

$w_0 = (z_0 \bmod (n-1)) + 1$ with n being the order n of base point G as defined for NIST P-256

$w_1 = (z_1 \bmod (n-1)) + 1$ with n being the order n of base point G as defined for NIST P-256

w_0 and w_1 shall not equal.

Then, it shall calculate:

$L = w_1 \times G$

with G as defined for the chosen elliptic curve.

The script algorithm is described in [\[11\]](#).

The Scrypt function is defined with the following parameters:

- **Salt:** 16 bytes, randomly generated for each new verifier as per [\[12\]](#)
- **Cost parameter Nscript:** 4096 or higher
- **Block size r:** 8
- **Parallelization parameter p:** 1
- **Output length dkLen:** 80

The Vehicle OEM Server then provides salt, L, and w0, which constitute the verifier, securely and confidentially to the vehicle.

This specification does not specify the means to upgrade these parameters to faster hardware in the future. A simple solution would be for the server to provide these parameters to the vehicle too.

When the pairing starts or re-starts after an erroneous attempt, the vehicle shall randomly generate a valid scalar y on the chosen curve, and then calculate:

$$Y = y \times G + w0 \times N$$

where N is a point on the used elliptic curve as specified in Section [18.1.5](#). When the maximum number of failed pairing attempts is reached (see the usage of SPAKE2+ REQUEST command in Section [5.1.2](#)), a new password and verifier shall be generated to allow to retry pairing.

Y shall not be the point at infinity and shall be on the chosen curve. A new y shall be generated, and a new Y shall be calculated until the requirements for Y are fulfilled before continuing the protocol.

The vehicle shall transmit Y to the device together with the Scrypt configuration parameters.

When owner pairing is started, the password is provided to the device to be paired, which allows it to calculate:

$$z0 = \text{left 40 bytes of Scrypt(pwd, s, Nscript, r, p, dkLen)}$$

$$z1 = \text{right 40 bytes of Scrypt(pwd, s, Nscript, r, p, dkLen)}$$

using s, Nscript, r, and p transmitted by the vehicle. dkLen is implicitly known on both sides.

It shall derive w0 and w1 using the method FIPS 186-4, B.5.1 Per-Message Secret Number Generation Using Extra Random Bits:

$$w0 = (z0 \bmod (n-1)) + 1 \text{ with } n \text{ being the order } n \text{ of base point } G \text{ as defined for NIST P-256}$$

$$w1 = (z1 \bmod (n-1)) + 1 \text{ with } n \text{ being the order } n \text{ of base point } G \text{ as defined for NIST P-256}$$

Then the device shall randomly generate a valid scalar x on the chosen curve and then calculate:

$$X = x \times G + w0 \times M$$

where M is a point on the chosen elliptic curve as specified in Section [18.1.5](#). A new x shall be generated and X shall be calculated on each new pairing attempt, within the limits of a maximum number of retries until a new password needs to be generated by the server.

X shall not be the point at infinity and shall be on the chosen curve. A new x shall be generated, and a new X shall be calculated until the requirements for X are fulfilled before continuing the protocol.

Device and vehicle then shall exchange X and Y through the commands defined in Section 5.

On reception of X, the vehicle shall compute the shared secret curve points Z and V as follows:

$$Z = y \times (X - w_0 \times M)$$
$$V = y \times L$$

On reception of Y, the device shall compute the shared secret curve points Z and V as follows:

$$Z = x \times (Y - w_0 \times N)$$
$$V = w_1 \times (Y - w_0 \times N)$$

Both vehicle and device then shall calculate the shared secret K as follows:

$$K = \text{SHA-256}(\text{len}(X) \parallel X \parallel \text{len}(Y) \parallel Y \parallel \text{len}(Z) \parallel Z \parallel \text{len}(V) \parallel V \parallel \text{len}(w_0) \parallel w_0)$$

where $\text{len}(\text{str})$ denotes the length of a string in bytes, represented as an 8-byte little-endian number.

The shared secrets CK and SK are derived from K:

$$\text{CK} = K[0:128]$$
$$\text{SK} = K[128:128]_8$$

The LONG_TERM_SHARED_SECRET is derived based on [Listing 18-9](#).

18.1.3 Verification

The verification of the derived keys shall be done by calculating a check value on either side which are mutually exchanged and verified.

The vehicle shall calculate the verification value M[1] as

$$K_1 = \text{HKDF}(\text{CK}, \text{"ConfirmationKeys"} \parallel \text{TLV } 5B_h \parallel \text{TLV } 5C_h, [0:128]), \text{ where}$$

ConfirmationKeys is a defined static string (see [Listing 18-6](#)).

$$M[1] = \text{C-MAC}(K_1, X) \text{ using } K_1 \text{ as secret key, using CMAC-AES-128 as defined in [RFC4493]}$$

and shall send it over to the device.

The device shall verify M[1] using the received M[1] from the vehicle and, if successful, shall calculate the verification value M[2] as

$$K_2 = \text{HKDF}(\text{CK}, \text{"ConfirmationKeys"} \parallel \text{TLV } 5B_h \parallel \text{TLV } 5C_h, [128:128]) \text{ (see } \textcolor{green}{\text{Listing 18-6}} \text{)}$$
$$M[2] = \text{C-MAC}(K_2, Y) \text{ using } K_2 \text{ as secret key, using CMAC-AES-128 as defined in [RFC4493]}$$

and shall send it over to the vehicle.

The protocol succeeds when the vehicle successfully validates M[2].

18.1.4 Summary

To summarize, the following elements are part of the protocol:

- s (16 bytes): Salt generated by the Vehicle OEM Server

⁸ The notation used is [start_index: number_of_bits]

- 1 • **pwd**: Password generated by the Vehicle OEM Server, available in the owner's Vehicle
- 2 OEM account
- 3 • **z0** (40 bytes): Part of verifier including FIPS extra bits, derived from the password using a
- 4 one-way function
- 5 • **z1** (40 bytes): Value including FIPS extra bits derived from the password using a one-way
- 6 function
- 7 • **w0** (32 bytes): Part of verifier, derived z0
- 8 • **w1** (32 bytes): Value derived from z1
- 9 • **L**: curve point, part of verifier, calculated by the Vehicle OEM Server in uncompressed
- 10 format
- 11 • **x** (32 bytes): Random scalar generated by device on each pairing attempt
- 12 • **y** (32 bytes): Random scalar generated by vehicle on each pairing attempt
- 13 • **M**: Known constant curve point in uncompressed format
- 14 • **N**: Known constant curve point in uncompressed format
- 15 • **X**: Intermediate public curve point exchanged between device and vehicle in uncompressed
- 16 format
- 17 • **Y**: Intermediate public curve point exchanged between vehicle and device in uncompressed
- 18 format
- 19 • **Z**: Secret curve point generated on both sides from public values and constants in
- 20 uncompressed format
- 21 • **V**: Secret curve point generated on both sides from public values and constants in
- 22 uncompressed format
- 23 • **K** (32 bytes): Shared SPAKE2+ secret
- 24 • **CK** (16 bytes): Shared SPAKE2+ secret (as derived from K) to derive confirmation keys
- 25 • **SK** (16 bytes): Shared SPAKE2+ secret (as derived from K) to derive system keys
- 26 • **K1** (16 bytes): Vehicle-side secret key for evidence calculation M[1]
- 27 • **K2** (16 bytes): Device-side secret key for evidence calculation M[2]
- 28 • **Kenc, Kmac, Krmac**: Secure channel keys
- 29 • **LONG_TERM_SHARED_SECRET**: Long-term shared secret for further key derivations

30 For all operations, all curve points shall be represented in x9.63 standard format as the byte
31 stream 0x04 || <x> || <y> format where <x> and <y> are each 32-byte integer in big-endian
32 representation (65 bytes).

33 x and y shall be newly generated after each failing pairing attempt.

34 18.1.5 SPAKE2+ Constant Definitions

35 The following NIST P-256 (SECP256r1) curve parameters are defined in [10]

36 M=

37 04

38 88 6e 2f 97 ac e4 6e 55 ba 9d d7 24 25 79 f2 99

1 3b 64 e1 6e f3 dc ab 95 af d4 97 33 3d 8f a1 2f
2 5f f3 55 16 3e 43 ce 22 4e 0b 0e 65 ff 02 ac 8e
3 5c 7b e0 94 19 c7 85 e0 ca 54 7d 55 a1 2e 2d 20
4
5 N=
6 04
7 d8 bb d6 c6 39 c6 29 37 b0 4d 99 7f 38 c3 77 07
8 19 c6 29 d7 01 4d 49 a2 4b 4f 98 ba a1 29 2b 49
9 07 d6 0a a6 bf ad e4 50 08 a6 36 33 7f 51 68 c6
10 4d 9b d3 60 34 80 8c d5 64 49 0b 1e 65 6e db e7

11 **18.2 Privacy Properties**

12 **18.2.1 NFC Transaction [WCC1/WCC2]**

13 The vehicle identifier is transmitted from the vehicle to the device in the AUTH0 command (see
14 Section [15.3.2.9](#)). It is recommended to change the vehicle identifier when the vehicle owner
15 changes to avoid any privacy issues.

16 **18.2.2 Vehicle OEM Server**

17 The Vehicle OEM Server shall be designed in a way to separate business-relevant data from the
18 key tracking data, which may be stored in a KTS for example.

19 Business data, such as number of purchased mobile keys, expiration of the mobile key
20 subscription, etc., may be exploited to enforce the Vehicle OEM business policy.

21 Data stored in the KTS shall be privacy protected. The data shall not be used for business
22 reasons.

23 When a vehicle is stolen (or in similar cases), only the KTS-data for the affected vehicle is to be
24 disclosed, on specific request.

25 **18.2.3 User Privacy**

26 The Vehicle OEM-controlled data going into the Digital Key instance shall be chosen and
27 protected in a way such that the privacy of device users is protected in all use cases.

28 **18.2.4 Multi-Vehicle OEM Deployment**

29 The device shall enforce that a specific Vehicle OEM application can only access Digital Key
30 data and functionality within the Digital Key applet instance that belongs to the same Vehicle
31 OEM.

32 **18.2.5 Error Handling**

33 On every attempt to execute the owner pairing flow, the device shall generate a new public key
34 pair (device.PK/device.SK).

18.3 Cryptographic Algorithms

All cryptographic elements that are stored in or received from the Digital Key applet shall respect the formats defined in Section 4.

18.4 Cryptographic Protocols

18.4.1 Server Password Generation

The Server generates the password for vehicle and device in the following way:

Listing 18-1: Server Password Generation

```
input: Script parameters
output: w0, w1, L, s, pwd
begin
  generate random salt s
  generate random password pwd
  z=Script(pwd, s) with parameters as defined in Section 18.1.2
  z0=left 40 bytes of z
  z1=right 40 bytes of z
  w0=(z0 mod (n-1)) + 1 with n being the order n of base point G as defined for NIST P-256
  w1=(z1 mod (n-1)) + 1 with n being the order n of base point G as defined for NIST P-256
  L=w1×G with G as defined for NIST P-256
  provide pwd to device (through web interface or Vehicle OEM app)
  send s, w0 and L securely to vehicle (through Vehicle OEM proprietary link)
end
```

18.4.2 Vehicle-side public EC Point Generation

Listing 18-2: Vehicle-side Public Point Generation

```
input: w0
output: Y, y
bBegin
  generate random scalar y on chosen curve
  Y=y×G+w0×N
return Y, which is sent to the device during owner pairing
end
```

18.4.3 Device-side public EC Point Generation

Listing 18-3: Device-side Public Point Generation

```
input: w0
output: X, x
begin
  generate random scalar x on chosen curve
  X=x×G+w0×M
return X, which is sent to the vehicle during owner pairing
end
```

1 18.4.4 Vehicle-side computation of Shared Secret

2 Listing 18-4: Vehicle-side Computation of Shared Secret

```
1 input: X, w0, L, y
2 output: K
3 begin
4   Z=y×(X-w0×M)
5   V=y×L
6   K=SHA-256(len(X) || X || len(Y) || Y || len(Z) || Z || len(V) || V || len(w0) || w0)
7   where len(str) denotes the length of a string in bytes, represented as an 8-byte little-endian number.
8   OKM : [0:128] = CK, [128:128] = SK (all 16-byte)
9 end
```

3 18.4.5 Device-side computation of Shared Secret

4 Listing 18-5: Device-side Computation of Shared Secret

```
1 input: Y, w0, w1, x
2 output: CK, SK
3 begin
4   Z=x×(Y-w0×N)
5   V=w1×(Y-w0×N)
6   K=SHA-256(len(X) || X || len(Y) || Y || len(Z) || Z || len(V) || V || len(w0) || w0)
7   where len(str) denotes the length of a string in bytes, represented as an 8-byte little-endian number.
8   OKM : [0:128] = CK, [128:128] = SK (all 16-byte)
9 end
```

5 18.4.6 Derivation of Evidence Keys

6 Listing 18-6: Derivation of Evidence Keys

```
1 input: CK
2 output: K1, K2
3 begin
4   Compute the steps indicated by IETF RFC5869 [13] with the following mapping:
5   IKM : CK
6   L : 32
7   Salt : NULL
8   Info : "ConfirmationKeys" || TLV 5Bh || TLV 5Ch see Table 5-4
9   Both TLVs include tag, length and data field.
10  Hash : SHA-256
11  OKM : [0:128] = K1, [128:128] = K2 (all 16-byte)
12  Notation is [start_index: number_of_bits]
13 end
```

7 18.4.7 Vehicle-side computation of Evidence

8 Listing 18-7: Vehicle-side Computation of Evidence

```
1 input: K1, X
2 output: M[1]
3 begin
4   compute M[1] = CMAC(K1, X) using K1 as secret key, using CMAC-AES-128 as defined in [RFC4493]
5   return M[1]
6 end
```


18.4.8 Device-side computation of Evidence

Listing 18-8: Device-side Computation of Evidence

```
1 input: K2, Y
2 output: M[2]
3 begin
4   compute M[2] = CMAC(K2, Y) using K2 as secret key, using CMAC-AES-128 as defined in [RFC4493]
5   return M[2]
6 end
```

18.4.9 Derivation of System Keys [WCC1/WCC2/WCC3]

Derivation of system keys Kenc, Kmac, Krmac for secure channel and of a long-term symmetric secret for further key derivations. If the vehicle is WCC2 or WCC3, and if the device is either WCC2 or WCC3 capable or device is capable of deriving Kble_intro and Kble_oob_master, then additionally Kble_intro and Kble_oob_master shall be derived. This algorithm is not part of SPAKE2+ but of the applicative part of the system.

Listing 18-9: Derivation of System Keys

```
1 input: SK, Flag_Kble_intro_Kble_oob_master_support
2 output: Kenc, Kmac, Krmac, LONG_TERM_SHARED_SECRET
3 If (Flag_Kble_intro_Kble_oob_master_support == true): Additional output: Kble_intro, Kble_oob_master
4 begin
5   Compute the steps indicated by IETF RFC5869 [13] with the following mapping:
6   L : if (Flag_Kble_intro_Kble_oob_master_support == true) 96 else 64
7   Salt : NULL
8   Info : "SystemKeys"
9   Hash : SHA-256
10  OKM : [0:128] = Kenc, [128:128] = Kmac, [256:128] = Krmac, [384:128] = LONG_TERM_SHARED_SECRET
      (all 16-byte)
      if (Flag_Kble_intro_Kble_oob_master_support == true): [512, 128] = Kble_intro; [640,128] = Kble_oob_master
      (all 16 bytes)
11  Notation is [start_index: number_of_bits]
12 end
```

18.4.10 Generate Attestation Signature

The signature is calculated with ECDSA according to [8] using NIST P-256 curve and SHA-256. See Listing 15-43.

18.4.11 Validate Attestation or Certificate

See Listing 15-44.

18.4.12 Secure Channel Command Encryption and Authentication

Listing 18-10: Secure Channel Command Encryption and Authentication

```
1 input: payload, Kmac, Kenc
2 output: encrypted_payload, mac
3 begin
4   Compute the steps as indicated in Section 6.2.6 of GPC_SPE_014 [7] with the following mapping:
5   Command Data Field - Plain Text : payload
6   Padded Counter Block : 00000000000000000000000000000000 || 1-byte counter (01h for first command, up to max FFh)
```

```

7   S-ENC : Kenc
8   CIPHERED Command Data Field : encrypted_payload
9   MACChainingValue : 00000000000000000000000000000000h on first command or
10      MAC Chaining value of previous command
11   S-MAC : Kmac
12   C-MAC : mac (8 bytes)
13   end

```

18.4.13 Secure Channel Response Encryption and Authentication

Listing 18-11: Secure Channel Response Encryption and Authentication

```

1   input: payload, Krmac, Kenc
2   output: encrypted_payload, mac
3   begin
4       Compute the steps indicated in Section 6.2.7 of GPC_SPE_014 [7] with the following mapping:
5       Response Data Field - Plain Text : payload
6       Padded Counter Block: 80000000000000000000000000000000h ||
7       1-byte counter_value used in command
8       S-ENC : Kenc
9       CIPHERED Response Data Field : encrypted_payload
10      Response Data Field : encrypted_payload
11      MAC Chaining Value : 16-byte MAC Chaining Value from command
12      S-RMAC : Krmac
13      R-MAC : mac (8 bytes)
14   end

```

18.5 Error Handling [WCC1/WCC2/WCC3]

Vehicle and device should limit the time between starting of the pairing mode and the start of the contactless transaction. After expiration of this time, the vehicle and device need to be brought back to pairing mode by the user.

The overall time for the pairing transaction should be limited on vehicle and device side. When the vehicle and device are brought into pairing mode and the pairing transaction starts (i.e., successful SELECT command/response exchange), the transaction time should be limited. Then, either the pairing was successful, or the pairing process will be aborted by the vehicle and device and needs to be restarted by the user. A new pairing password shall be entered again on the device side.

The time to present the device again after a RF reset should be limited. The vehicle should abort the pairing process if a device is not presented within this time limit after the restart of the reader field.

18.6 Secure Element (SE)

Secure Element (SE): As defined within CCC, is embedded technology that provides a tamper-resistant secure implementation aimed at providing isolation, integrity, confidentiality, secure provisioning, secure storage, and secure processing for the Digital Key applet including its high-value keying material while being isolated from the rich OS and application processor. An implementation shall provide hardware and software protection against physical and logical attacks and unsafe use during the full lifecycle of the secure implementation, the Digital Key applet, and their assets.

1 **Tamper-resistant secure hardware:** Hardware designed to isolate and protect embedded
2 software and data by implementing appropriate security measures. The hardware and embedded
3 software must be resistant against high attack potential and must meet the requirements as
4 defined by the Digital Key certification which include resistance to physical tampering scenarios.
5 The acceptable scenarios are described in the latest Security IC Platform Protection Profile or in
6 other alternatives identified by the Digital Key certification.

7 The transport channel between a SE and the NFC component shall protect confidentiality and
8 integrity. The SE shall provide a secure binding between these components that shall be resistant
9 to manipulation after production. This is referred to as a hardware root of trust. It shall be
10 prevented that any code running on the application processor has access to or inject data on this
11 channel. The Secure Element shall have the ability to distinguish communication between wired
12 interface and contactless interface.

19 BLUETOOTH LOW ENERGY (LE) INTERFACE [WCC2/WCC3]

The Secure Ranging Service provides the Bluetooth LE framework necessary to discover, manage, and control UWB-based ranging between a device and a vehicle. Bluetooth LE, Secure Element, and UWB are core to the Digital Key solution. Bluetooth LE offers secure data exchange between device and vehicle which then enables Secure Element to offer mutual authentication and data sharing over a secure channel with the Vehicle. The Bluetooth LE channel is also required to establish and or manage the Secure Ranging service.

19.1 Bluetooth LE Functional Requirements

The Bluetooth LE Controller and Host for both the device and vehicle shall be compliant to the mandatory capabilities of the Bluetooth Core Specification 5.0 or later [30] and shall support the following additional capabilities:

- LE L2CAP Connection Oriented Channel Support
- LE Privacy
- LE Secure Connections

The Bluetooth LE Controller and Host for both the device and vehicle may optionally support the following additional capabilities:

- Long Range (LELR)
- LE Advertising Extensions (mandatory, if LE Long Range is supported)

Vehicles and devices shall use “LE security mode 1/Level 4” or/and “Secure Connections Only mode,” see Vol 3, Part C, Section 10 of [30]. During a connection, the device and vehicle may use the Feature Exchange Procedure to signal the LELR support based on the “LE Coded PHY” bit defined in [30].

19.1.1 Vehicle

The Vehicle shall support the GAP Peripheral role.

The vehicle shall support privacy in the advertising state as defined in Volume 6, Part B, Section 6 of [30]. The vehicle shall use resolvable private addresses for advertising events as specified in Volume 6, Part B, Section 6.2 of [30].

If the vehicle supports multiple Bluetooth LE controllers, all controllers shall use the same identity address and resolution key. Each controller may have a different random resolvable address at any time.

The vehicle shall support GATT in the server role.

19.1.2 Device

The device shall support the GAP Central role.

The device shall support resolvable private addresses used in advertisements by the vehicle.

- 1 The device shall initiate a connection to the vehicle when it receives the advertising packet from
- 2 a paired vehicle. A connection interval of 30ms is recommended when connecting to a vehicle.
- 3 The device shall support GATT in the client role.

4 **19.2 Bluetooth LE Procedures**

5 *19.2.1 Owner Pairing Connection Establishment*

6 This is the flow for owner pairing connection establishment using Bluetooth out-of-band (OOB)

7 pairing. The Bluetooth LE Setup is divided into the following two subsections:

- 8 • Bluetooth LE Link Layer Connection Establishment & Bluetooth LE Owner Pairing GATT
- 9 Flow
- 10 • Bluetooth LE Pairing & Encryption Setup

11 A vehicle that does not support secure ranging shall not allow owner pairing over the Bluetooth

12 LE interface and shall not broadcast the Owner Pairing Advertisement. Owner pairing shall start

13 over NFC as described in Section 6 and followed by Bluetooth LE Activation flow as described

14 in Section [19.5.9.5](#).

15 *19.2.1.1 Bluetooth LE Link Layer Connection Establishment & Bluetooth LE Owner Pairing*

16 *GATT Flow*

17 In [Figure 19-1](#), the vehicle begins by sending ADV_IND with CCC_DK_UUID as the

18 advertising payload (steps 11 and 15). The Vehicle LL shall be in the advertising state with its

19 filtering policy set to accept all connection requests. The device begins passive scanning. The

20 Device LL shall be in the scanning state with its filter policy set to accept all advertisements.

21 Once the Device LL receives an advertisement, it forwards that to the device host. The Device

22 Host shall check if the CCC_DK_UUID is contained in the advertisement payload (box A). If the

23 CCC_DK_UUID is contained in the advertising payload, the user is notified. If the user accepts

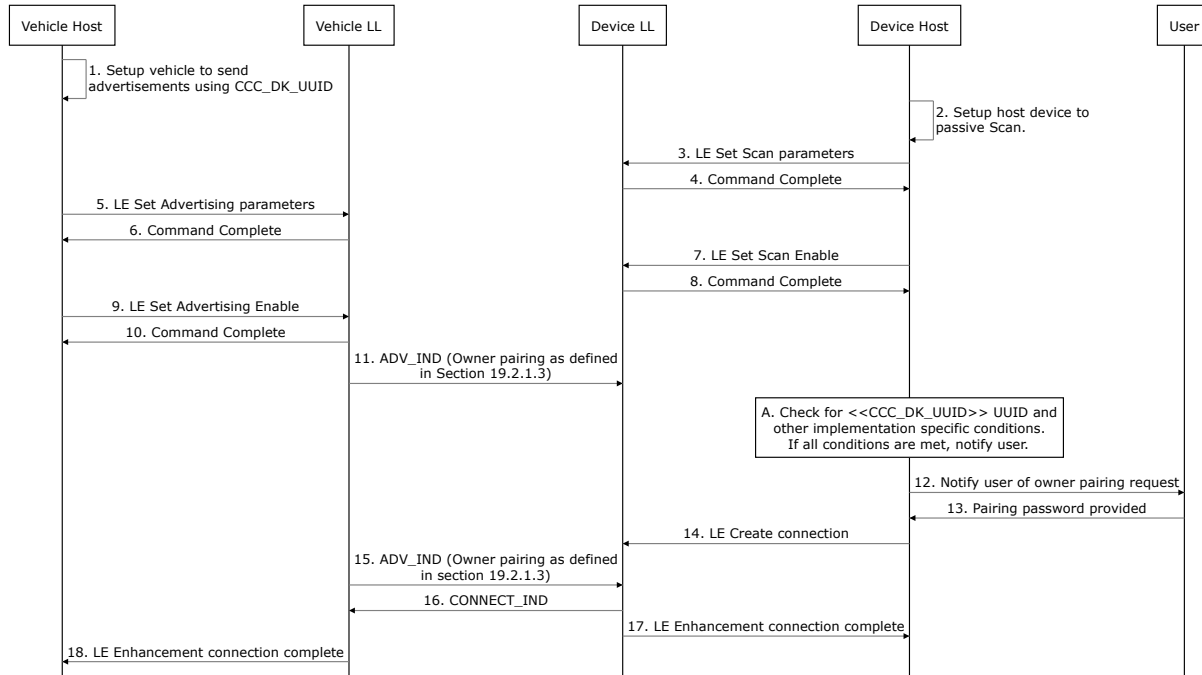
24 the owner pairing requests, the user provides the pairing password. The Device LL shall enter

25 initiating state with the filter policy set to the advertiser's address after step 14. Upon receiving

26 the next same advertisement from the Vehicle LL, the Device LL shall send a connection request

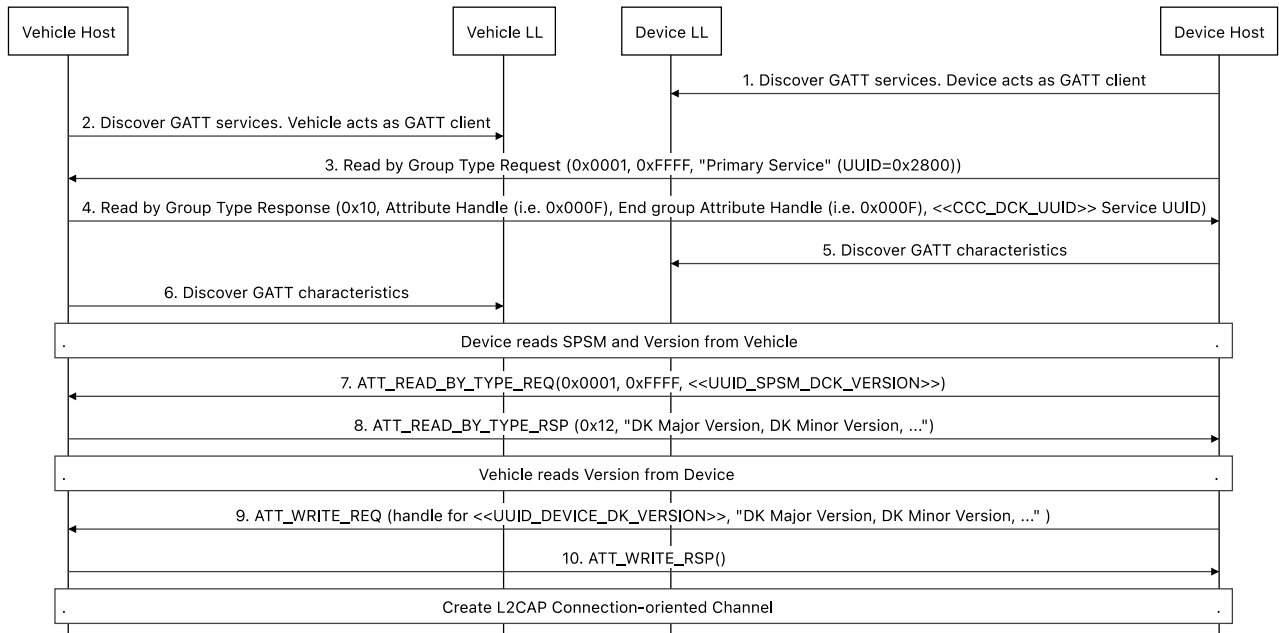
27 (CONNECT_IND as in step 16).

1 *Figure 19-1: Bluetooth LE Link Layer Connection Establishment.*



2
3 In [Figure 19-2](#), the Device Host (acting as the GATT client) initiates service discovery with the
4 Vehicle Host (acting as the GATT server) to get the DK Service and DK Service UUID_SPSM
5 DK_VERSION characteristic in order to establish the L2CAP connection for the DK Service.

6 *Figure 19-2: L2CAP Connection-Oriented Channel.*

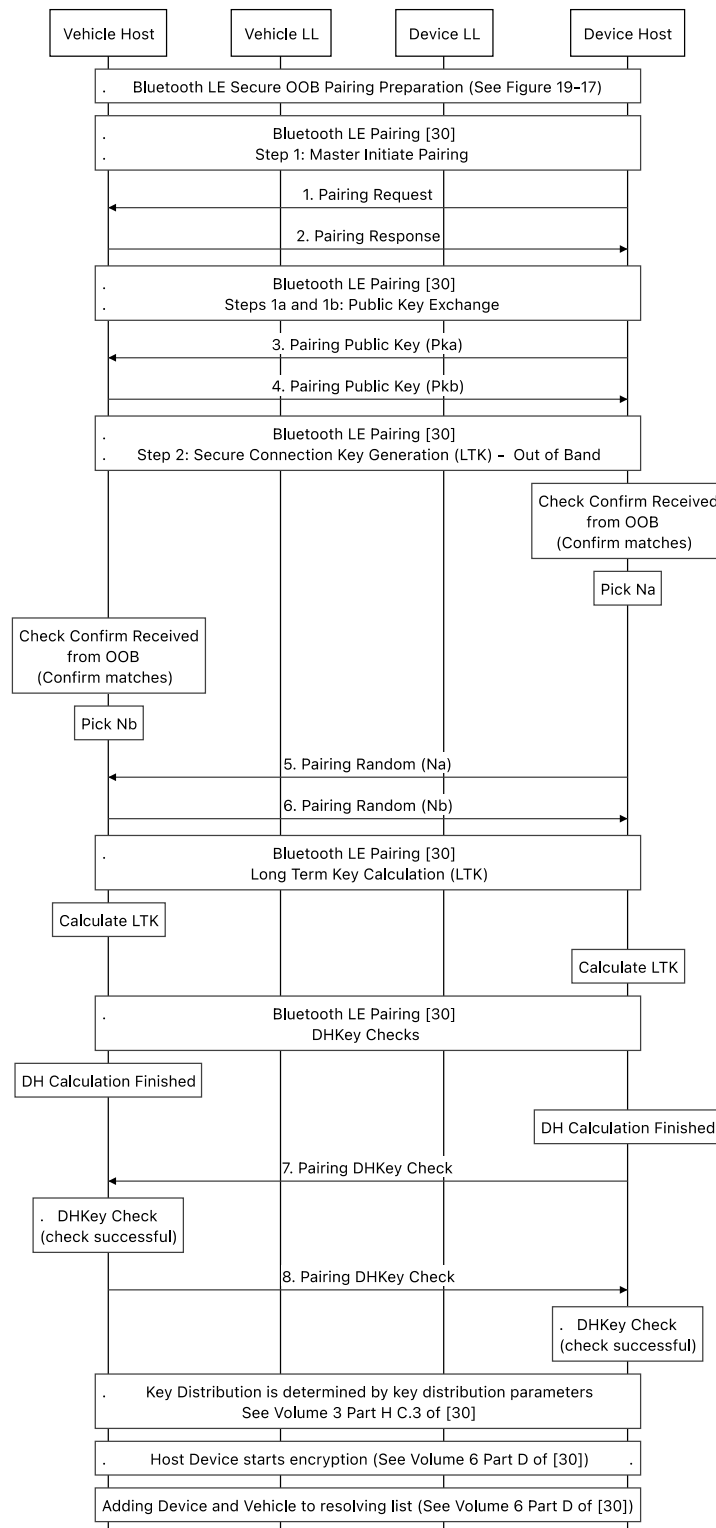


7
8 [Figure 19-2](#) is an example in which the device reads the SPSM value from the vehicle's GATT
9 Server. The device may read the SPSM value in other ways, as defined in the Volume 3 Part G
10 of [\[30\]](#).

- 1 For more details on Box B and the associated message flow, please refer to Volume 3 Part A
- 2 [\[30\]](#).

1 19.2.1.2 Bluetooth LE Pairing and Encryption Setup

2 *Figure 19-3: Bluetooth LE Pairing and Encryption Setup.*



3

In [Figure 19-3](#), the setup begins with the Bluetooth LE Secure OOB Pairing Preparation procedure as defined in Section [19.5.1](#) and shown in Figure 19-16.

During this procedure the device has generated its public key (PKa) and secret key (SKa) and has received the vehicle's Bluetooth address (BTAddrB), the vehicle's commitment value (Cb), and random number (rb) over a secured channel. The vehicle has generated its public key (PKb) and secret key (Skb) and received the device's Bluetooth address (BTAddrA) and the device's commitment value (Ca) and random number (ra) over a secured channel.

The procedure continues with the following:

- **Bluetooth LE Pairing: Master Initiated Pairing** (Pairing Phase 1, Step 1) as defined in [\[30\]](#). The Bluetooth LE Pairing Request PDU is sent from the device to the vehicle. The Bluetooth LE Pairing Response PDU is sent from the vehicle to the device.
- **Bluetooth LE Pairing: Public Key Exchange** (Pairing Phase 1, step 1a and 1b) Vol 2, Part H, Section 7.1 in [\[30\]](#). A Bluetooth LE Pairing Public Key PDU is sent from the device to the vehicle and a Bluetooth LE Pairing Public Key PDU is sent from the vehicle to the device. Both the device and vehicle begin their DHKey generation.
- **Bluetooth LE Pairing: Secure Connection Key Generation** (LTK) – OOB (Pairing Phase 2, Authentication Stage 1, Step 2) in [\[30\]](#). Both the device and vehicle verify if the confirm value received as part of the OOB Pairing Preparation procedure matches. Both the device and vehicle generate a random nonce (Na and Nb). The device sends its nonce (Na) to the vehicle using the Bluetooth LE Pairing Random PDU and the vehicle sends its nonce (Nb) to the device using the Bluetooth LE Pairing Random PDU.
- **Bluetooth LE Pairing: LTK calculation** (Pairing Phase 2, Authentication Stage 2) as described in Vol 3, Part H, Section 2.3.5.6.5 in [\[30\]](#). Once the DHKey generation is completed on the device and vehicle, the device and vehicle calculate their LTK.
- **Bluetooth LE Pairing: DHKey Checks**. The device sends the check value (Ea) using the Pairing DHKey Check PDU to the vehicle. The vehicle sends the check value (Eb) using the Pairing DHKey Check PDU to the device. Both the device and vehicle verify the values.
- **Bluetooth LE Pairing: Key Distribution** (Pairing Phase 3) as described in Vol 3, Part H, Section 2.4.3.2 in [\[30\]](#).
- **Device and vehicle enabling encryption** as described in Vol 3, Part H, Section 2.4.4.2 in [\[30\]](#). The device and vehicle add each other to their private address resolving list as described in Vol 6, Part B, Section 4.7 in [\[30\]](#).

The messages in the box “Host Device starts encryption” and “Adding device and vehicle to resolving list” are out of scope of this specification. These are shown for illustration purposes only. Please refer to Vol 6 Part D of [\[30\]](#).

19.2.1.3 Owner Pairing Advertising

For owner pairing, only the Legacy LE 1M PHY shall be used. The Advertisement (ADV_IND) for owner pairing shall follow the Section 2.3.1.1 of Volume 6 Part B of [\[30\]](#).

Event Type: Connectable and scannable undirected

The ADV_IND consists of Advertising Address and Advertising Data as shown in [Table 19-1](#) and [Table 19-2](#), respectively:

Table 19-1: AdvA field of ADV_IND.

Field	Length (bytes)
Advertising Address (AdvA)	6

Table 19-2: AdvData field of ADV_IND.

Field	Length (bytes)	Description	Value (hex)
Length	1	Length of AD Type and AD Data	0x03
AD Type	1	16-bit service UUID	0x03
AD Data	2	<<CCC_DK_UUID>>	0xFFFF5
Length	1	Length of AD Type and AD Data	0x14
AD Type	1	Service data -128bit UUID	0x21
AD Data	16	CCCSERVICEDataIntent UUID	0x5810bbc0-b499-11e9-a2a3-2a2ae2dbcc4
AD Data	1	IntentConfiguration	0x1 (Default)
AD Data	2	Vehicle Brand Identifier	AS defined in Table 2-1 of [35]

The usage of CCC_DK_UUID: The Assigned Value is provided by Bluetooth SIG, Inc. and may only be used by its members in compliance with all terms and conditions of use issued by Bluetooth SIG, Inc. For more information visit <https://www.bluetooth.com/specifications/assigned-numbers>.

The IntentConfiguration is used to allow separate user flows on the device, depending on whether the user has started owner pairing in the vehicle or not. The byte is defined as described in Table 19-3

Table 19-3: Definition of IntentConfiguration byte.

Field	Description	Bit index
Intent	0: Intent from vehicle - user has actively started owner pairing in vehicle 1: No intent from vehicle	0
Reserved for future use	0: Set to 0 and shall be ignored by receiver	1:7

The vehicle brand identifier is specified in Table 2-1 in [35].

19.2.1.4 Pairing Request Definition:

The following fields are mandatory for the Pairing Request (see Volume 3, Part H, Section 3.5.1 of [30]):

Table 19-4: Mandatory fields in Pairing Request.

Field	Value (hex)	Definition
IO Capability	0x00 – 0xFF	See Vol 3, Part H, Table 3.4 in Section 3.5 of [30]
OOB data flag	1	OOB Authentication data from remote device present

Field	Value (hex)	Definition
AuthReq	1	Bonding

19.2.1.5 Pairing response definition:

The following fields are mandatory for the Pairing Response (see Volume 3, Part H, Section 3.5.2 of [30]):

Table 19-5: Mandatory fields in Pairing Response.

Field	Value (hex)	Definition
IO Capability	0x00 – 0xFF	See Vol 3, Part H, Table 3.4 in Section 3.5 of [30]
OOB data flag	0x1	OOB Authentication data from remote device present
AuthReq	0x1	Bonding
Maximum Encryption Key Size	0x10	
Initiator Key Distribution	0xF0	
Responder Key Distribution	0xF0	

19.2.1.6 DK Service:

The vehicle shall support the DK Service. This shall be a primary service and there shall only be one instance of this service.

Table 19-6: DK Service UUID.

Field	Value
DK Service UUID	<<CCC_DK_UUID>>

19.2.1.7 Vehicle PSM Characteristic:

This characteristic shall return the Simplified Protocol/Service Multiplexer (SPSM) used for the L2CAP channel from the vehicle.

The device shall read the SPSM from the vehicle upon connection. The vehicle manufacturer shall pick an SPSM. For LE Credit-based Connections, a dynamically allocated SPSM value (i.e., the non-SIG assigned SPSM value in 0x0080-0x00FF) is used.

During passive entry, First New Key Transaction and owner pairing, the device shall upon Bluetooth LE connection:

- Read the characteristics by UUID (with the UUID_SPSM or UUID_SPSM_DK_VERSION). The SPSM value shall use big-endian byte order.
- Write the characteristics by UUID (for UUID_DEVICE_DK_VERSION)
- Bluetooth Encryption Requirements are described in Section 19.2.2.

1 *Table 19-7: SPSM Characteristic declaration.*

Attribute type	Attribute value		Attribute permission
0x2803 – UUID for «Characteristic»	Characteristic Properties = 0x02	0xMMMM = Handle of Characteristic Value D3B5A130-9E23-4B3A-8BE4- 6B1EE5F980A3 – UUID_SPSM	Read only, No Authentication, No Authentication

2 *Table 19-8: SPSM Characteristic value declaration.*

Attribute type	Attribute value	Attribute permission
D3B5A130-9E23-4B3A-8BE4-6B1EE5F980A3 – UUID_SPSM	uint16 – SPSM value	Read only, No Authentication, No Authentication

3 19.2.1.8 DK Version Characteristic:

4 *Table 19-9: Vehicle SPSM and DK version Characteristic Declaration*

Attribute type	Attribute value		Attribute permission
0x2803 – UUID for «Characteristic»	Characteristic Properties = 0x02	0xMMMM =Handle of Characteristic Value AE285B91-6D23-23F1-CA12- 6B1EE5B780A3 – UUID_SPSM_DK_VERSION	Read only, Authenticated Encryption required (for passive entry)

5 *Table 19-10: Vehicle SPSM and DK Version Characteristic Value Declaration*

Attribute type	Attribute value	Attribute permission
AE285B91-6D23-23F1-CA12- 6B1EE5B780A3 – UUID_SPSM_DK_VERSION	Refer to Table 19-13 (Attribute value definition for Vehicle_SPSM_And_DK_Version_Characteristic)	Read only, Authenticated Encryption required (for passive entry)

6 *Table 19-11: Device Selected DK Version Characteristic Declaration*

Attribute type	Attribute value		Attribute permission
0x2803 – UUID for «Characteristic»	Characteristic Properties = 0x02	0xMMMM = Handle of Characteristic Value BD4B9502-3F54-11EC-B919- 0242AC120005 – UUID_DEVICE_DK_VERSION	Write only, Authenticated, Encryption required (for passive entry)

7 *Table 19-12: Device Selected DK Version Characteristic Value Declaration*

Attribute type	Attribute value	Attribute permission
BD4B9502-3F54-11EC-B919- 0242AC120005 – UUID_DEVICE_DK_VERSION	Refer to Table 19-14 (Attribute value definition for Device_Selected_DK_Version_Characteristic)	Write only, Authenticated, Encryption required (for passive entry)

Table 19-13: Attribute Value Definition for Vehicle SPSM and DK Version Characteristic

Attribute value	Length (Bytes)	Description
SPSM value	2	SPSM Value.
Supported_DK_Protocol_Version_Len	1	Length byte for supported DK Protocol versions.
Supported_DK_Protocol_Version	2 × n	DK_Protocol_Version is 2 Byte field (Major:Minor); minor version byte changes whenever there is a change to DK protocol flows or message definitions. Major version byte changes when backward compatibility is broken. The vehicle shall return all supported DK Protocol Version using two bytes per version in big endian coding. n is the number of protocol versions supported by the vehicle The DK Protocol Version shall be ordered from the highest to the lowest version. In this specification the vehicle shall support the DK Protocol Version 3.0 (coded 0300_h). The device shall select the highest DK Protocol Version commonly supported by device and vehicle as described in Selected_DK_Protocol_Version attribute (See Table 19-14) The first supported DK Protocol Version is 0100_h.
Features Supported length	1	1 byte is the length in bytes for the Features Supported Field per DK protocol version
Features Supported	k	k is the number of bytes needed to convey the bitfield for the list of features supported. Refer to Table 19-15.

Table 19-14: Device Selected DK Version Characteristic

Attribute value	Length (Bytes)	Description
Selected_DK_Protocol_Version	2	Device-selected highest DK Ranging Protocol Version commonly supported by device and vehicle. In this specification, the device shall support the DK Ranging

		Protocol Version 3.0 (coded 0300h). Attribute may not be included if Supported_DK_Ranging_Protocol_Version_Len = 0x00. Refer to Table 19-13 .
Features Supported length	1	Length in bytes for the Features Supported Field
Features Supported	k	k is the number of bytes needed to convey the bitfield for the list of features supported in common by vehicle and device. Refer to Table 19-15.

The device shall respond with the latest version supported. DK Protocol Major Version and DK Protocol Minor Version attribute values shall be encoded as big endian.

DK Protocol Major Version may not be backward compatible with previous DK Protocol Major Versions.

DK Protocol Minor Version shall be compatible with previous DK Protocol Minor Version for the same DK Protocol Major Version.

If UUID_SPSM_DK_VERSION is not available on vehicle, it shall be interpreted that the vehicle supports DK Protocol Version 1.0 (i.e., DK Protocol Major Version is 1 and DK Protocol Minor Version is 0).

If the device supports DK Protocol version 1.0, it shall request the UUID_SPSM from the vehicle.

If the device supports DK Protocol Version higher than 1.0:

- Device shall request the UUID_SPSM_DK_VERSION characteristic from the vehicle. If the vehicle doesn't support "UUID_SPSM_DK_VERSION", the device shall ask for "UUID_SPSM" since the vehicle only supports DK Protocol version 1.0.
- Device shall use the highest version commonly supported by both vehicle and device.
- If vehicle supports V-D-BT version higher than 1.0, device shall write the Selected_DK_Protocol_Version and Features_Supported to the Vehicle UUID_DEVICE_DK_VERSION characteristic.

If the vehicle supports DK Protocol Version 1.0, it shall implement UUID_SPSM. If the vehicle that does not support UUID_SPSM_DK_VERSION and the UUID_DEVICE_DK_VERSION characteristic receives a request for the UUID_SPSM_DK_VERSION or UUID_DEVICE_DK_VERSION from a device, it shall respond with the error code "Attribute not found" as indicated by Bluetooth specification (refer to Bluetooth core specification v5.0 Vol 3, Part F 3.4.4.1).

If the vehicle supports DK Protocol Version higher than 1.0:

- Vehicle shall implement the UUID_SPSM and the UUID_SPSM_DK_VERSION characteristic

The total size of the attribute value for UUID_SPSM and UUID_SPSM_DK_Version shall be less than 512 bytes. If this value is exceeded, it shall keep the most recent anchor versions followed by additional supported versions to make it fit within 512 Bytes.

Features Supported definition:

Features Supported shall be defined as follow:

Table 19-15: Supported Feature Definition Bitmap

Attribute type/ Feature Supported	Attribute value	Description
Bit[0]	0 or 1	1 _b : TimeSync Procedure 0 Supported Optional for vehicle, See 19.4.4.1
Bit[1]	0 or 1	1 _b : TimeSync Procedure 1 Supported Optional for vehicle, See 19.4.4.2
Bit[2]	0 or 1	1 _b : URSK derivation via Standard Transaction for other purposes Optional for device and vehicle, See 19.5.6.1
Bit[3]	0 or 1	1 _b : Head Unit Pairing Optional for device and vehicle, See 19.5.2
Bit[4]	0 or 1	1 _b : LE Coded PHY Passive Entry Advertising Optional for device and vehicle, See 19.2.3.1
Bit [5]	0 or 1	1 _b : Encrypted UWB Final Data over BLE Optional for device and vehicle. See 19.3.6
Bit [6]	0 or 1	1 _b : PDR Data Supported 0 _b : PDR Data Not Supported PDR Data support is optional for device and for vehicle

19.2.1.9 Vehicle Bluetooth Tx Power Level Characteristic:

The Vehicle Bluetooth Tx Power Level characteristic is an optional attribute provided by the vehicle. The Vehicle Bluetooth Tx Power Level characteristic shall return the current radiated transmit power of a Bluetooth LE module in dBm [see GATT specification, supplement version 4].

Table 19-16 Vehicle Bluetooth Tx Power Level Characteristics

Attribute type	Data Type	Size (in octets)
0x2A07 – UUID for «Tx Power Level»	sint8	1

The device should read the Vehicle Bluetooth Tx Power level from the vehicle upon connection. During passive-entry, the device should read the characteristics by UUID (with the UUID for “Tx power level”) at reconnection. The Bluetooth LE module on the vehicle should keep the same radiated transmit power during the entire time of the Bluetooth LE connection.

19.2.1.10Vehicle Antenna Identifier Characteristic:

The Vehicle antenna identifier represents a unique identifier for every physical Bluetooth LE antenna located in the vehicle that establishes a connection to a device. The antenna identifier is defined as a unique value that correspond to a unique physical Bluetooth antenna in the vehicle. This shall remain a unique antenna identifier even if a different device connects to that antenna. The device should read the “Vehicle Antenna Identifier” from the vehicle upon connection. In the case a Bluetooth module has multiple antennas, it can be considered as single unique single antenna. Also, if multiple Bluetooth modules share multiple antennas, then it can also be considered as a single antenna.

The GATT characteristic instance should be unique per client connecting to the vehicle if the vehicle supports multiple concurrent Bluetooth LE connections.

During passive-entry, the device should read the characteristics by UUID (with the UUID_AntennaIdentifier) at reconnection.

Table 19-17: Vehicle Antenna Characteristic declaration

Attribute type	Attribute value			Attribute permission
0x2803 – UUID for «Characteristic»	Characteristic Properties = 0x02	0xMMMM = Handle of Characteristic Value	c6d7d4a1-e2b0-4e95-b576-df983d1a5d9f – UUID_AntennaIdentifier	Read only, No Authentication, No Authorization

Table 19-18: Vehicle Antenna Characteristic value declaration

Attribute type	Attribute value	Attribute permission
c6d7d4a1-e2b0-4e95-b576-df983d1a5d9f – UUID_ AntennaIdentifier	Uint16 – Vehicle Antenna Identifier	Read only, No Authentication, No Authorization

19.2.2 Bluetooth Encryption

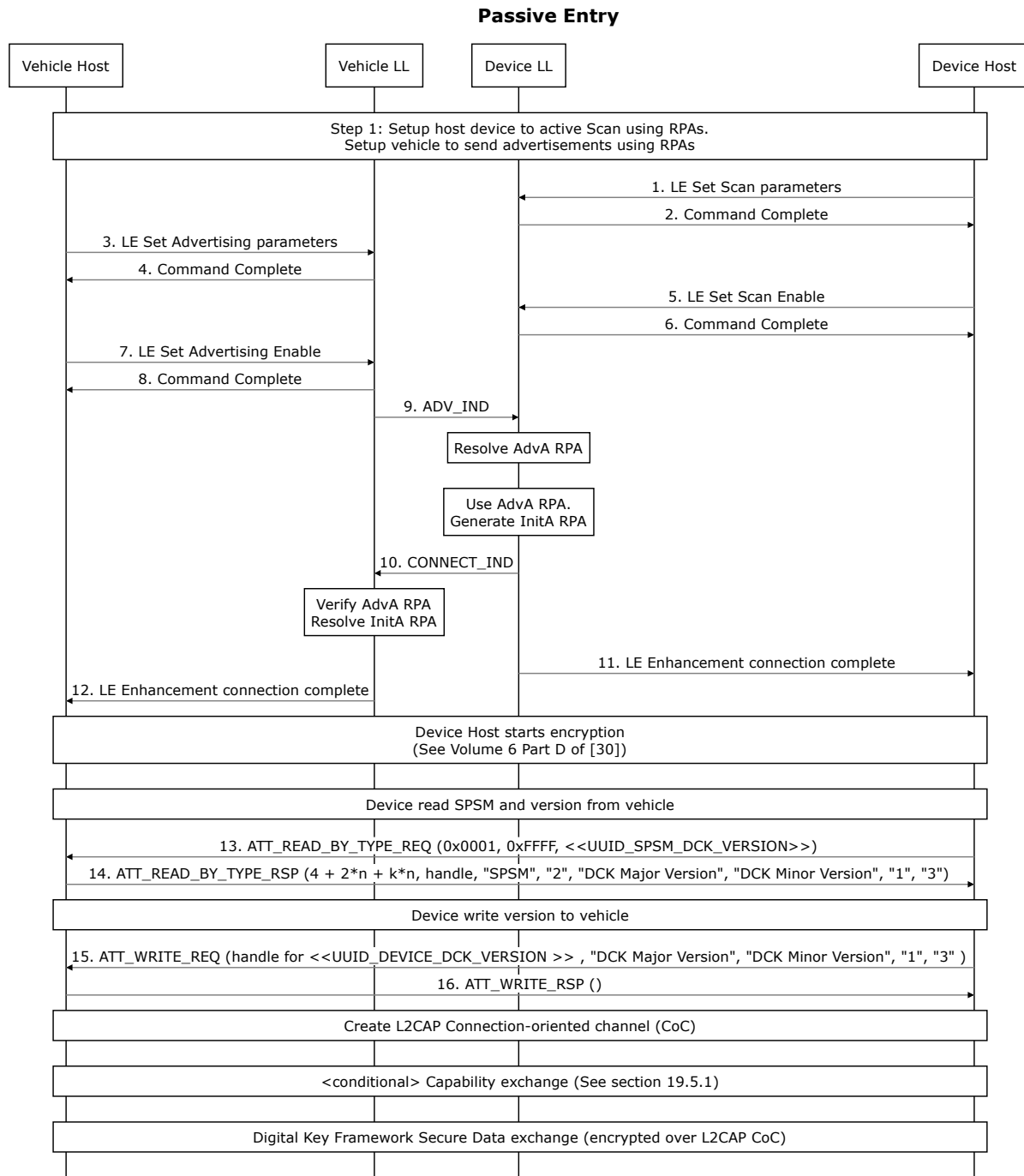
The following requirements apply to Bluetooth encryption:

- The device (central) shall request encryption and encryption shall successfully complete before L2CAP connection establishment is initiated, except for owner pairing and First New Key Transaction.
- The vehicle shall trigger a disconnect 5 seconds after an unencrypted L2CAP connection establishment if no First_Approach_RQ (see Section 19.3.4.1) or Request_owner_pairing Command Complete SubEvent notification (see Table 19-73) has been received during the first approach and owner pairing, respectively.
- During owner pairing and first new key transaction, the vehicle shall trigger a disconnect if the device does not enable encryption 5 seconds after successful completion of LE pairing phase 2.
- During owner pairing and first new key transaction, the vehicle shall trigger a disconnect if the device does not enable encryption 5 seconds after successful completion of LE pairing phase 3.
- The device and vehicle should not terminate the L2CAP connection used for DK Service while the BLE connection is alive. In case the L2CAP connection used for DK Service is terminated while the Bluetooth LE connection is alive, the device shall attempt to re-establish the DK Service L2CAP connection once conditions permit.

19.2.3 Passive Entry: Bluetooth LE Setup:

Once the Bluetooth LE Pairing and Encryption of the Bluetooth LE setup is completed, the subsequent connection to the vehicle shall use the passive entry flow.

Figure 19-4: Passive Entry Flow Diagram.



For passive entry, Capability Exchange is conditional upon whether either the vehicle or device has an updated DK Protocol Version, UWB Configuration Identifier, PulseShape Combinations, or a combination of these parameters since the last Capability Exchange, as described in Section 19.5.1.

19.2.3.1 Passive Entry Advertising:

The vehicle shall advertise on the LE 1M PHY.

When advertising on the LE 1M PHY, the connectable and scannable undirected event shall be used, see Volume 6, Part B, Section 4.4.2.7 in [30]. The advertising interval shall be 42.5ms.

The vehicle may advertise on both the LE Coded PHY with S=2 coding and on the LE 1M PHY. If both PHYs are used, the advertising for each shall occur consecutively.

When advertising on the LE Coded PHY, the connectable undirected event shall be used, see Volume 6, Part B, Section 4.4.2.3 in [30]. The advertising interval shall be 84ms.

For both advertisements, the ADV_IND shall contain AdvA but no advertising data (AdvData).

A paired device shall be able to connect to both the owner pairing advertising and the passive entry advertising to perform the passive entry flows. For known devices connecting to passive entry as well as owner pairing advertisement, the vehicle shall be able to perform the Passive Entry Flow.

19.2.3.2 Passive Entry L2CAP Connection

Upon establishment of the LE link layer connection, the device host shall establish an L2CAP connection using the protocol specified in Section 19.2.1.1.

19.2.3.3 Connection Performance Recommendations

In this section, the connection performance parameters and their values are recommended. The exact values are refined and finalized in the certification/test plan specification, which form the certification requirement. All TBD values in this section will be finalized/measured during interoperability event.

- **Device requirement:**

- During the approach of the user toward the vehicle, the last Bluetooth message before the UWB pre-poll should be sent by 3 (TBD_1) meters for 95 (TBD_2) percentiles. This shall be based on the following assumptions:

- The maximum approach walk speed is 2.1m/s and the test start at 6 (TBD_3) meters.
- The conditions defined in certification/test plan specification.
- Device starts Bluetooth LE scanning at the start of test.

- **Transmitter requirements:**

- Bluetooth LE Advertisement packet (from vehicle), measured with reference measurement setup, at 10 meters (TBD_5) should be equal or greater than -80 dBm (TBD_4) with a free-space LOS path loss and common test case condition (see note below).
- The radiated power output power level (including antenna gain) of a CONNECT_IND packet from mobile device should be equal or greater than -80 dBm (TBD_6) at 10 meters (TBD_5) with a free-space LOS path loss and common test case condition (see note below).

Note: Reference measurement setup and common test case condition are defined in certification/test plan specification.

• **Receiver requirements:**

The actual sensitivity level, as defined in Version 5.2 Vol 6 Part A Section 4 of [30]:

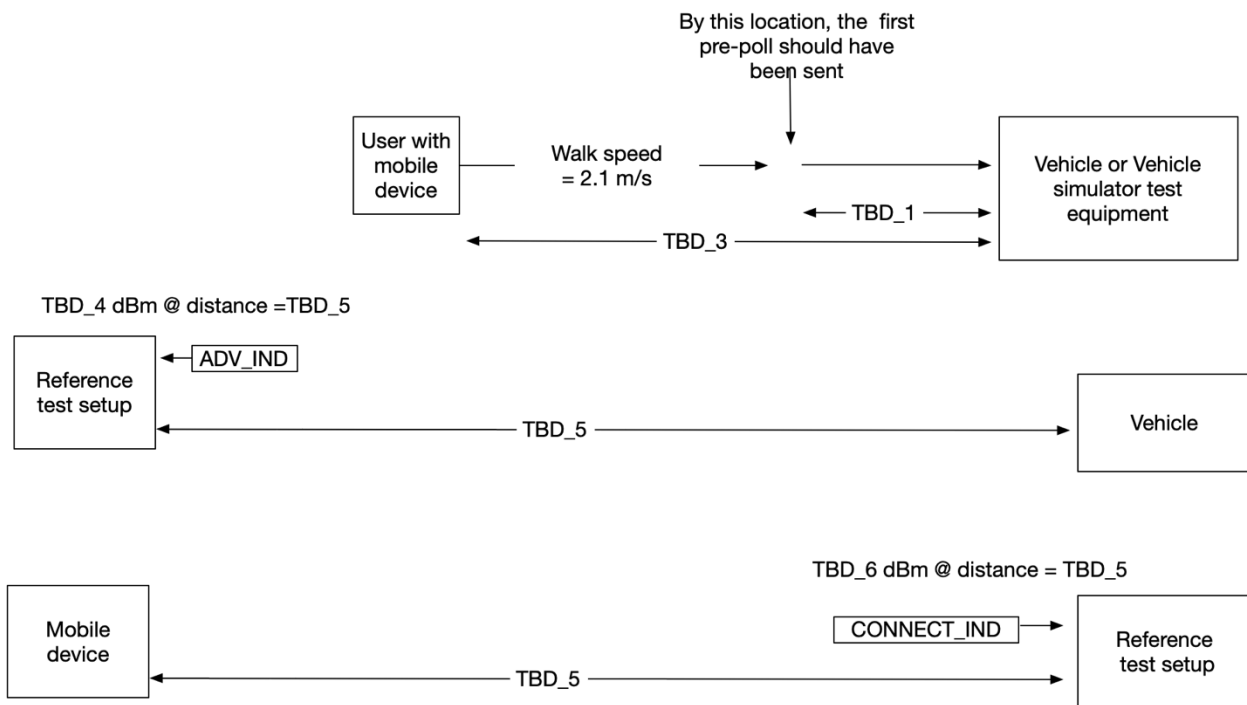
- Should be equal or better than -85dBm for un-coded PHYs and -88dBm for LE Coded PHY with S=2 coding for the device (see note below).
- Should be equal or better than -85dBm for un-coded PHYs and -88dBm for LE Coded PHY with S=2 coding for the vehicle (see note below).

Note: Reference measurement setup and common test case condition are defined in certification/test plan specification.

• **General requirements:**

Test should be done using the optimal ranging setup flow with a previously derived URSK
Any other losses in the Bluetooth link such as vehicle body or interior should be kept at a minimum or at best avoided.

Figure 19-5: Connection performance in relationship with device, transmitter, and receiver requirements.



19.2.3.4 Vehicle Requirements

Maximum advertising intervals:

Maximum number of advertising events per time:

Locked vehicles shall transmit within 504ms (6 periods of 84ms) up to a maximum of

- (a) 24 LE 1M + 12 LELR or
- (b) 36 LE 1M + 6 LELR or
- (c) 48 LE 1M (zero LELR)

advertising events (See Section 19.2.3.1).

Unlocked vehicles shall transmit within 504ms (6 periods of 84ms) up to a maximum of

- (a) 48 LE 1M + 24 LELR or
- (b) 72 LE 1M + 12 LELR or
- (c) 96 LE 1M (zero LELR)

advertising events (See Section [19.2.3.1](#)).

19.3 DK Message Format

The DK messages shall be exchanged over L2CAP using the DK Service SPSM.

The device shall retrieve the DK Service SPSM from the vehicle's GATT server and establish an LE L2CAP credit-based connection to the vehicle's SPSM for use by the DK Protocol (See [Figure 19-2](#)).

The DK message format is shown in [Table 19-19](#).

Table 19-19: DK Message Format.

Message Field	Location
Message Header	Byte 0
Payload Header	Byte 1
Length	Byte [3:2]
Data	N

The Message Header field is one byte long and indicates the Message Type of the message being sent. For example, if the message being sent is a Select APDU wrapped within DK_APDU_RQ (see Section [19.3.2.1](#) for more details), the Message Header field indicates whether this message is an SE message for standard transaction or Framework message for owner pairing, for which the associated message processing differs by the type of message. On the other hand, the Payload Header field is 1 byte long and communicates the message such as DK_APDU_RQ.

The Message Header is defined in [Table 19-20](#) and the Message Type is defined in [Table 19-21](#). Bit [7:6] of the Message Header are reserved for future use and shall be set to 0.

Table 19-20: Message Header definition.

Message Header	Bit field
Message Type	Bit [5:0]
RFU	Bit [7:6]

Table 19-21: Message Type definition.

Message Type definition Bit [5:0]	Value (decimal)
Framework message	0
SE message	1
UWB Ranging Service message	2

Message Type definition Bit [5:0]	Value (decimal)
DK Event Notification	3
Vehicle OEM App message	4
Supplementary Service message	5
Head Unit Pairing message	6
Reserved	7–63

1 The Payload Header field is defined in Table 19-22:

2 *Table 19-22: Payload Header definition.*

Payload Header	Bit field
Message ID	Bit [7:0]

3 The Length field is two bytes long and indicates the size of the Data field in the message coded
4 in Big Endian format. All the DK message fields shall be encoded in Big Endian format, except
5 for data type fields already defined by Bluetooth Specification or otherwise explicitly defined in
6 this specification. Those data types ordering defined by Bluetooth Specification shall remain
7 unchanged.

8 The Data field is variable in length and its data structure definition is defined in Sections [19.3.1](#)
9 to [19.3.9](#) for each message ID.

10 Table 19-23 shows the categorization of DK message under different Message Type.

11 *Table 19-23: Message Type and its associated messages.*

Message Type	Message	APDUs
Framework	DK_APDU_RQ DK_APDU_RS (As defined in Section 19.3.2)	SELECT, SPAKE2+ REQUEST, SPAKE2+ VERIFIER, WRITE DATA, GET DATA, GET RESPONSE, OP CONTROL FLOW, See Table 5-1
SE	DK_APDU_RQ DK_APDU_RS (As defined in Section 19.3.2)	SELECT, AUTH0, AUTH1, EXCHANGE CONTROL FLOW CREATE RANGING KEY See Table 15-1
UWB Ranging Service	Ranging_Capability_RQ Ranging_Capability_RS Ranging_Session_RQ	N/A

Message Type	Message	APDUs
	Ranging_Session_RS Ranging_Session_Setup_RQ Ranging_Session_Setup_RS Ranging_Suspend_RQ Ranging_Suspend_RS Ranging_Recovery_RQ Ranging_Recovery_RS Configurable_Ranging_Recovery_RQ Configurable_Ranging_Recovery_RS (As defined in Section 19.3.1)	
DK Event Notification	Ranging_Event (As defined in Section 19.3.9)	N/A
Vehicle OEM App	Pass_Through (As defined in Section 19.3.6)	N/A
Supplementary Service	Time_Sync (As defined in Section 19.3.3) First_Approach_RQ First_Approach_RS (As defined in Section 19.3.4) RKE_Auth_RQ RKE_Auth_RS (As defined in Section 19.3.5) UWB_Final_Data (As defined in Section 19.3.6)	N/A
Head Unit Pairing	Head_Unit_Pairing_Preparation Head_Unit_Pairing_RQ Head_Unit_Pairing_RS (As defined in Section 19.3.8)	N/A

- 1 Below is an example on how to encode an APDU message before transporting it over Bluetooth
- 2 LE by using DK Message. The same can also be used to decode the incoming message.

```

3 Message: Select APDU message (See Section 15.3.2.1) over Bluetooth LE with
4 AID as 0xA000000809434343444B417631.
5 Message Type in Message Header: 0x01 (SE)
6 Message ID in Payload Header: 0x0B (DK_APDU_RQ, 11)
7 Data: 0x00A404000DA000000809434343444B41763100
8 (SELECT APDU command)
9 DK Bluetooth LE Payload: 0x010B001300A404000DA000000809434343444B41763100

```

19.3.1 UWB Ranging Service Message

This section details the Ranging Service messages to negotiate, initiate and complete UWB ranging between the device and the vehicle.

19.3.1.1 Ranging Capability Request (RC-RQ)

This message is sent by the vehicle to the device during ranging capability exchange. The Ranging Capability Request message and its parameters are defined in Table 19-24 and Table 19-25, respectively.

Table 19-24: Ranging_Capability_RQ message and its parameters.

Message	Message ID	Parameters
Ranging_Capability_RQ	0x01	Supported_DK_Protocol_Version_Len, Supported_DK_Protocol_Version, Supported_UWB_Config_Id_Len, Supported_UWB_Config_Id Supported_PulseShape_Combo_Len Supported_PulseShape_Combo

Table 19-25: Definition of the parameters for Ranging_Capability_RQ.

Parameters	Length (bytes)	Description
Supported_DK_Protocol_Version_Len	1	Length byte for supported DK protocol versions
Supported_DK_Protocol_Version	2 × n	DK_Protocol_Version is 2 Byte field (Major:Minor); as defined in Table 19-13. If a DK_Protocol_Version was selected as part of the GATT version negotiation, this field shall contain only the selected DK_Protocol_Version. The first supported DK Protocol Version is 0100 _h .
Supported_UWB_Config_Id_Len	1	Length byte for supported UWB Configuration Identifiers
Supported_UWB_Config_Id	2 × m	UWB_Config_Id is a 2 Byte field which is an identifier for the supported UWB configuration. This configuration includes the supported PHY layer parameters. See Section 21.4 for the supported UWB PHY layer configurations. m is the number of UWB configs supported by vehicle
Supported_PulseShape_Combo_Len	1	Length byte for the supported PulseShape_Combos
Supported_PulseShape_Combo	1 × l	PulseShape_Combo is a 1 Byte field which is an identifier for the supported initiator/response, transmit/receive pulse shape combinations. See Section 21.5 for the supported subset. l is the number of PulseShape_Combos supported by the vehicle.

19.3.1.2 Ranging Capability Response (RC-RS)

The message is sent by the device to the vehicle in response to a Ranging Capability Request message (RC-RQ) during ranging capability exchange. The Ranging Capability Response message and its parameters are defined in Table 19-26 and Table 19-27, respectively.

Table 19-26: Ranging_Capability_RS message and its parameters.

Message	Message ID	Parameters
Ranging_Capability_RS	0x02	Selected_DK_Protocol_Version,

Message	Message ID	Parameters
		Selected_UWB_Config_Id Selected_PulseShape_Combo

1 *Table 19-27: Definition of the parameters for Ranging_Capability_RS.*

Parameters	Length (bytes)	Description
Selected_DK_Protocol_Version	2	Highest DK Protocol Version commonly supported by device and vehicle defined in Table 19-14 . The first supported DK Protocol Version is 0100h. For DK Protocol Version higher than version 1.0, please refer to Section 19.2.1.7. If supported, the DK Protocol Version selected during GATT negotiation shall be used by device.
Selected_UWB_Config_Id	2	Device-selected UWB config id from the list of supported common UWB config ids between device and vehicle.
Selected_PulseShape_Combo	1	Device-selected single pulse shape combination from the list of common supported ones between device and vehicle.

2 19.3.1.3 Ranging Session Request (RS-RQ)

3 This message is sent by the vehicle to the device to start the ranging session setup handshake
4 procedures as defined in Section [20.5.2](#). The Ranging_Session_RQ message and its parameters
5 are defined in Table 19-28 and Table 19-29, respectively

6 The vehicle shall use the Selected_DK_Protocol_Version, Selected_UWB_Config_id and
7 Selected_PulsShape_Combo as selected by the device during ranging capability exchange (see
8 Figure 19-17).

9 If the vehicle has an updated DK Protocol Version or UWB Configuration Identifier or
10 PulseShape Combinations or combination of these parameters, the vehicle shall perform
11 Capability Exchange with the device again as described in Section [19.5.1](#).

12 *Table 19-28: Ranging_Session_RQ message and its parameters.*

Message	Message ID	Parameters
Ranging_Session_RQ	0x03	Selected_DK_Protocol_Version, Selected_UWB_Config_id, UWB_Session_Id, Selected_PulseShape_Combo, Channel_Bitmask

13 *Table 19-29: Definition of the parameters for Ranging_Session_RQ.*

Parameters	Length (bytes)	Description
Selected_DK_Protocol_Version	2	Selected protocol version from ranging capability exchange (RC-RQ, RC-RS)
Selected_UWB_Config_Id	2	Selected UWB config id from ranging capability exchange (RC-RQ, RC-RS)
UWB_Session_Id	4	The least significant 4 bytes of the transaction_identifier field. For more details on transaction_identifier, See Table 15-29. Vehicle selects the transaction_identifier based on which pre-derived URSK it chooses to use for establishing UWB ranging session. The least significant 4 bytes of that transaction_identifier

Parameters	Length (bytes)	Description
		is then used as UWB_Session_Id. Device uses UWB_Session_Id for URSK retrieval and session management.
Selected_PulseShape_Combo	1	Device-selected single pulse shape combination from the list of common supported ones between device and vehicle.
Channel_Bitmask	1	A bitmap equaling the bitwise “OR” of supported UWB channels Bit 0: UWB channel 5 supported Bit 1: UWB channel 9 supported Bit 2: RFU (shall be set to 0) Bit 3: RFU (shall be set to 0) Bit 4: RFU (shall be set to 0) Bit 5: RFU (shall be set to 0) Bit 6: RFU (shall be set to 0) Bit 7: RFU (shall be set to 0)

19.3.1.4 Ranging Session Response (RS-RS)

This message is sent by the device according to Section 20.5.2. The Ranging Session Response message and its parameters are defined in Table 19-30 and Table 19-31, respectively. If the device has an updated DK Protocol Version or UWB_Config_Id or PulseShape Combinations or combination of these parameters which is different than the ones it received in Ranging Session Request, the device shall respond with DK Event Notification with Subevent_Category set to 0x01_h and the corresponding command status code set to 0x02_h, to initiate Capability Exchange as described in Section 19.5.1.

Table 19-30: Ranging_Session_RS message and its parameters.

Message	Message ID	Parameters
Ranging_Session_RS	0x04	RAN_Multiplier, Slot_BitMask, SYNC_Code_Index_BitMask, Selected_UWB_Channel, Hopping_Config_Bitmask

Table 19-31: Definition of the parameters for Ranging_Session_RS.

Parameters	Length (bytes)	Value	Description
RAN_Multiplier	1	Range = 1 to 255 T_Block RAN = RAN_Multiplier × 96ms Time Range = 96ms to 24480 ms	RAN multiplier for setting the ranging block duration. See Section 20.5.2 for the definition ranging block.
Slot_BitMask	1	A bitmap equaling the bitwise “OR” combination of 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80.	Bitmap of supported values of Slot durations as a multiple of T _{Chap} , N _{Chap_per_Slot} as defined in Section 20.5.2. Each “1” in this bit map corresponds to a specific value of N _{Chap_per_Slot} where: 0x01 = “3” 0x02 = “4” 0x04 = “6” 0x08 = “8” 0x10 = “9”

Parameters	Length (bytes)	Value	Description
			0x20 = “12” 0x40 = “24” and 0x80 is reserved. Note that 0x40 is used for testing only
SYNC_Code_Index_BitMask	4	A bitmap equaling the bitwise “OR” combination of 0x00000001, 0x00000002, 0x00000004, ... 0x40000000, 0x80000000	Bitmap of SYNC code indices that can be used. The position of each “1” in this bit pattern corresponds to the index of a SYNC code that can be used, where: 0x00000001 = “1” 0x00000002 = “2” 0x00000004 = “3” 0x00000008 = “4” ... 0x40000000 = “31” 0x80000000 = “32” Refer to [31] and Section 21.4.1 for SYNC code index definition.
Selected_UWB_Channel	1	“0x01 for channel 5” and “0x02 for channel 9”	Selected UWB channel for requested session. See Section 21.7.1 .
Hopping_Config_Bitmask	1	0-255	[b7 b6 b5] bitmask of hopping modes, the device offers to use in this ranging session 100 No Hopping 010 Continuous Hopping 001 Adaptive Hopping The mandatory mode is 010 corresponding to Continuous Hopping mode. The support of 100 or 001 is optional. [b4 b3 b2 b1 b0] bit mask of hopping sequences the device offers to use in this ranging session. b4=1 is always set since the default hopping sequence and support thereof is mandatory and defined in Appendix A. b3=1 is set when the optional AES-based hopping sequence, which is defined in Appendix G, is supported. All remaining bits reserved for future hopping sequences.

19.3.1.5 Ranging Session Setup Request (RSS-RQ)

This message is sent by the vehicle as part of the ranging session setup handshake according to Section 20.5.2. The Ranging Session Setup Request message and its parameters are defined in Table 19-32 and Table 19-33, respectively.

Table 19-32: Ranging_Session_Setup_RQ message and its parameters.

Message	Message ID	Parameters
Ranging_Session_Setup_RQ	0x05	Session_RAN_Multiplier, Number_Chaps_per_Slot, Number_Responders_Nodes,

Message	Message ID	Parameters
		Number_Slots_per_Round, SYNC_Code_Index, Selected_Hopping_Config_Bitmask See Section 20 for the definition of these parameters.

1

Table 19-33: Definition of the parameters for Ranging_Session_Setup_RQ.

Parameters	Length (bytes)	Value	Description
Session_RAN_Multiplier	1	Range = 1 to 255 T_Block_S = Session_RAN_Multiplier × 96 ms Time Range = 96ms to 24480 ms	Session specific RAN multiplier. Value selected by vehicle must be greater than or equal to RAN_Multiplier sent by mobile device.
Number_Chaps_per_Slot	1	3, 4, 6, 8, 9, 12, or 24 Note that the last value is used only for testing	Selected Slot duration as a multiple of T _{Chap} , N _{Chap_per_Slot} as defined in Section 20.
Number_Responders_Nodes	1	1-255	Number of logical responder nodes participating in this ranging session as selected by the vehicle. See [31] for the definition of a logical node.
Number_Slots_per_Round	1	6,8,9,12,16,18,24,32,36,48,72, or 96	Selected number of slots per round
SYNC_Code_Index	4	A bitmap equaling the bitwise “OR” combination of 0x00000001, 0x00000002, 0x00000004, ... 0x40000000, 0x80000000	Bitmap of SYNC code indices that the vehicle can use (this may be a smaller set of the SYNC codes support by the vehicle). The position of each “1” in this bit pattern corresponds to the index of a SYNC code that can be used, where: 0x00000001 = “1” 0x00000002 = “2” 0x00000004 = “3” 0x00000008 = “4” ... 0x40000000 = “31” 0x80000000 = “32” See [31] and Section 21 for SYNC code index definition.
Selected_Hopping_Config - Bitmask	1	0-255	Vehicle shall send a bitmask indicating the selected hopping mode and the selected hopping sequence (in case of continuous or adaptive hopping). This bitmask shall meet the following rules: - For [b7 b6 b5], only one of those bits shall be set to 1 - For [b5 b4 b3 b2 b1 b0], at max one of the bits is set to 1 to signal the selected hopping sequence and the rest of the bits are set to 0. Examples of a valid Selected_Hopping_Config_BitMask [10000000] no hopping mode [01010000] continuous hopping with default hopping sequence [00110000] adaptive hopping with default hopping sequence

Parameters	Length (bytes)	Value	Description
			[01001000] continuous hopping with AES-based optional hopping sequence [00101000] adaptive hopping with AES-based optional hopping sequence

19.3.1.6 Ranging Session Setup Response (RSS-RS)

This message is sent by the device to the vehicle to complete the ranging session setup handshake according to Section 20.5.2. The Ranging Session Setup Response message and its parameters are defined in Table 19-34 and Table 19-35, respectively.

Table 19-34: Ranging_Session_Setup_RS message and its parameters.

Message	Message ID	Parameters
Ranging_Session_Setup_RS	0x06	STS_Index0, UWB_Time0, HOP_Mode_Key, SYNC_Code_Index See Section 20 for the definition of these parameters

Table 19-35: Definition of the parameters for Ranging_Session_Setup_RS.

Parameters	Length (bytes)	Value	Description
STS_Index0	4	0 - 0x3FFFFFFF	Starting STS index
UWB_Time0	8	0 - 0xFFFFFFFFFFFFFFFF	Starting time reference (resolution 1 microsecond) on UWB Device Clock of ranging session (see Section 20.3).
HOP_Mode_Key	4	0 - 0xFFFFFFFF	Key to generate default or AES based hopping sequence. In case of no hopping mode, this parameter shall be set to 0.
SYNC_Code_Index	1	1-32	Selected SYNC code index

19.3.1.7 Ranging Suspend Request Message (RSD-RQ)

This message is used to suspend an active ranging session for a given UWB_Session_Id. The Ranging Suspend Request message and its parameters are defined in Table 19-36 and Table 19-37, respectively.

Table 19-36: Ranging_Suspend_RQ message and its parameter.

Message	Message ID	Parameters
Ranging_Suspend_RQ	0x07	UWB_Session_Id

Table 19-37: Definition of the parameter for Ranging_Suspend_RQ.

Parameters	Length (bytes)	Description
UWB_Session_Id	4	The UWB Session ID of the currently active UWB ranging session.

19.3.1.8 Ranging Suspend Response Message (RSD-RS)

This message is used as a successful response to Ranging Suspend Request (RS-RQ). The Ranging Suspend Response message and its parameters are defined in Table 19-38 and Table 19-39, respectively.

Table 19-38: Ranging_Suspend_RS message and its parameter.

Message	Message ID	Parameters
Ranging_Suspend_RS	0x08	Suspend_Response

Table 19-39: Definition of the parameter for Ranging_Suspend_RS.

Parameters	Length (bytes)	Description
Suspend_Response	1	0x00 - For accepted suspend request 0x01 - To delay the suspend and keep ranging active

19.3.1.9 Ranging Recovery Request Message (RR-RQ)

This message is sent by vehicle to recover a suspended ranging session for a given UWB_Session_Id. The Ranging Recovery Request message and its parameters are defined in Table 19-40 and Table 19-41, respectively. The device and vehicle shall use the Session_RAN_Multiplier negotiated prior to ranging suspension (see Section 19.3.1.5).

Table 19-40: Ranging_Recovery_RQ message and its parameter.

Message	Message ID	Parameters
Ranging_Recovery_RQ	0x09	UWB_Session_Id

Table 19-41: Definition of the parameter for Ranging_Recovery_RQ.

Parameters	Length (bytes)	Description
UWB_Session_Id	4	The UWB Session ID of a previously suspended UWB ranging session.

19.3.1.10 Ranging Recovery Response Message (RR-RS)

This message is sent by device as a successful response to vehicle's Ranging Recovery Request (RR-RQ). The Ranging Recovery Response message and its parameters are defined in Table 19-42 and Table 19-43, respectively.

Table 19-42: Ranging_Recovery_RS message and its parameter.

Message	Message ID	Parameters
Ranging_Recovery_RS	0x0A	STS_Index0, UWB_Time0

Table 19-43: Definition of the parameter for Ranging_Recovery_RS.

Parameters	Length (bytes)	Description
STS_Index0	4	Starting STS index
UWB_Time0	8	Starting time reference (resolution 1 microsecond) on UWB Device Clock of ranging session (see Section 20.3).

19.3.1.11 Configurable Ranging Recovery Request Message (CRR-RQ)

This message is sent by vehicle to recover a suspended ranging session for a given UWB_Session_Id. The Configurable Ranging Recovery Request Message and its parameters are defined in Table 19-44 and Table 19-45, respectively. This message enables the vehicle to request a different RAN Multiplier value than the one selected prior to the suspension (i.e.

Session_RAN_Multiplier). For example, the vehicle may request an aggressive frequency to support higher ranging rate for critical use cases such as door handle touch sensor.

Table 19-44: Configurable_Ranging_Recovery_RQ message and its parameter.

Message	Message ID	Parameters
Configurable_Ranging_Recovery_RQ	0x12	UWB_Session_Id Requested_RAN_Multiplier

Table 19-45: Definition of the parameter for Configurable_Ranging_Recovery_RQ.

Parameters	Length (bytes)	Description
UWB_Session_Id	4	The UWB Session ID of the currently active UWB ranging session.
Requested_RAN_Multiplier	1	Requested RAN Multiplier for the ranging session to be recovered. The Requested_RAN_Multiplier shall be different than the Session_RAN_Multiplier prior to ranging suspension (see Section 19.3.1.5).

19.3.1.12 Configurable Ranging Recovery Response Message (CRR-RS)

This message is sent by device as a response to CRR-RQ. The Configurable Ranging Recovery Response message and its parameters are defined in Table 19-46 and Table 19-47, respectively.

Table 19-46: Configurable_Ranging_Recovery_Response message and its parameter.

Message	Message ID	Parameters
Configurable_Ranging_Recovery_RS	0x13	Selected_RAN_Multiplier STS_Index0, UWB_Time0

Table 19-47: Definition of the parameter for Configurable_Ranging_Recovery_Response.

Parameters	Length (bytes)	Description
Selected_RAN_Multiplier	1	Selected RAN Multiplier for the ranging session to be recovered. The Selected_RAN_Multiplier shall be equal or greater than the Requested_RAN_Multiplier
STS_Index0,	4	Starting STS index
UWB_Time0	8	Starting time reference (resolution 1 microsecond) on UWB Device Clock of ranging session (see Section 20.3).

The Selected_RAN_Multiplier shall be used during the recovered ranging session and shall remain valid until the ranging session ends or the ranging session is suspended. When the ranging session is recovered again, the RAN_Multiplier value for the given ranging session shall revert to the default Session_RAN_Multiplier for that ranging session.

19.3.2 SE Message

19.3.2.1 APDU Command Encapsulation

The DK_APDU_RQ message is an encapsulation of the APDU command specified as available on the wired interface in Table 15-1. The message and its parameters are defined in Table 19-48 and Table 19-49, respectively.

Table 19-48: DK_APDU_RQ message and its parameter.

Message	Message ID	Parameter
DK_APDU_RQ	0x0B	APDU command

Table 19-49: Definition of the parameter for DK_APDU_RQ.

Parameter	Length (bytes)	Description
APDU command	Variable	List of APDU commands are described in Table 19-23. Allowed class byte values are limited to 00 _h (for SELECT command), 80 _h (for commands with non-secure messaging) or 84 _h (for commands with secure messaging).

If the Digital Key framework has selected the Digital Key applet over a supplementary logical channel, it shall update the class byte of the APDU command accordingly before forwarding the APDU command to the Digital Key applet.

19.3.2.2 APDU Response Encapsulation

The DK_APDU_RS message is an encapsulation of the APDU response. Each DK_APDU_RQ message requires a corresponding DK_APDU_RS message. The message and its parameter are defined in Table 19-50 and Table 19-51, respectively.

Table 19-50: DK_APDU_RS message and its parameter.

Message	Message ID	Parameter
DK_APDU_RS	0x0C	APDU response

Table 19-51: Definition of the parameter for DK_APDU_RS.

Parameter	Length (bytes)	Description
APDU response	Variable	The APDU response corresponding to the APDU command

19.3.3 Supplementary Service Message - Time Sync

This message is used by device to provide Bluetooth LE Timesync payload which includes, the DeviceEventCount, the UWB Device Time timestamp and the UWB Device Time uncertainty.

The following two conditions shall trigger a timestamp (see Section 19.4):

- **Procedure 0:** After a CONNECT_IND, the device shall send a Time_Sync message
- **Procedure 1:** Once Bluetooth LE connection has been established, the vehicle may trigger a new Bluetooth LE event-based timestamp by requesting “LE Set PHY”. Device shall use the anchor point of the connection event used for “LL_PHY_UPDATE_IND” Link Layer message to generate a new time synchronization timestamp and send it to vehicle using this message.

Table 19-52 and Table 19-53 describe the parameters meaning based on which condition the timesync message was triggered.

Table 19-52: Time_Sync message and its parameters.

Message	Message ID	Parameters
Time_Sync	0x0D	DeviceEventCount,

Message	Message ID	Parameters
		UWB_Device_Time, UWB_Device_Time_Uncertainty, UWB_Clock_Skew_Measurement_available, Device_max_PPM, Success, RetryDelay

1

Table 19-53: Definition of the parameter for Time_Sync.

Parameters	Length (bytes)	Value	Description
DeviceEventCount	8	0 – 0xFFFFFFFFFFFFFFFE	Procedure 0: Event count is 0 or in case of retransmission, close to 0. Procedure 1: Event count value used for the LL_PHY_UPDATE_IND Link layer message. A value of 0xFFFFFFFFFFFFFFF indicates DeviceEventCount is unavailable on the device.
UWB_Device_Time	8	0 – 0xFFFFFFFFFFFFFFF	UWB Ranging clock value in microseconds. This is running clock and shall remain valid for the entirety of the session.
UWB_Device_Time_Uncertainty	1	0 – 0xFF	Time uncertainty between the anchor point of connection event and UWB Clock domain mapping. Format is log 2 accuracy of unsigned value in microseconds divided by 8: Value [µsec] = $2^{(UWB_Device_Time_Uncertainty / 8)}$ The value range which can be represented is 1µsec ($=2^0\mu\text{sec}$) to 1.094 hour ($=2^{255/8}\mu\text{sec}$)
UWB_Clock_Skew_Measurement_available	1	0 – 0x1	This is used to inform the vehicle if the device has a clock system which allows the vehicle to estimate and correct for the UWB clock skew, based on DeviceEventCount and UWB_Device_Time information. In particular this requires that the device UWB clock is available at Procedure 0 and Procedure 1. Value = 1: UWB clock skew measurement is available Value = 0: UWB clock skew measurement is NOT available.
Device_max_PPM	2	0-0xFFFF	Worst case clock skew of device UWB clock (with respect to nominal time). Format: Unsigned value representing (single sided) clock skew. Shall be interpreted that clock skew could be either positive or negative. Units is PPM (parts per million).
Success	1	0-0x2	0: The “Bluetooth LE Timesync” procedure has failed and the timesync message parameters are invalid. The vehicle should ask again to perform the “Bluetooth LE Timesync” after RetryDelay 1: The “Bluetooth LE Timesync” procedure on device side is successfully able to map the UWB clock and all parameters in the timesync message are valid. 2: The “Bluetooth LE Timesync” procedure has failed and the Procedure 0 is not available. This only applies to Procedure 0.

Parameters	Length (bytes)	Value	Description
RetryDelay	2	0-0xFFFF	Unsigned 16bits. Value in milliseconds for minimum delay required by device until the vehicle should trigger a new “Bluetooth LE Timesync”.

19.3.4 Supplementary Service Message - First Approach

First Approach messages enable Bluetooth LE OOB Secure LE pairing for both sender and receiver devices. These messages are used to share OOB Bluetooth LE data via application level encrypted channel through Kble_intro and Kble_oob as described in Bluetooth LE Secure OOB Pairing Prep (see Figure 19-16).

19.3.4.1 First Approach Request Message (FA-RQ)

This message is sent by the device to the vehicle to share device OOB data. The device OOB data comprises a combination of two encrypted payloads i.e. E1_Payload and E2_Payload. The E1_Payload and E2_Payload are encrypted using Kble_intro and Kble_oob, respectively. The message and its parameters are defined in Table 19-54 and Table 19-55, respectively.

Table 19-54: First_Approach_RQ message and its parameters.

Message	Message ID	Parameters
First_Approach_RQ	0x0E	E1_Payload IV1, Tag1, E2_Payload IV2, Tag2

Table 19-55: Definition of the parameter for First_Approach_RQ.

Parameters	Length (bytes)	Description
E1_Payload	8	E1_Payload is containing DK_Identifier. The value of E1_Payload is encrypted (AES-128-CCM) using Kble_intro as the encryption key. If slot_identifiers are provided by the vehicle, the DK_Identifier is an 8-byte zero padded slot_identifier based on the slot_identifiers provided by the vehicle. If slot identifiers are provided online and BLE key material is provided by the owner device (Kble_oob and Kble_intro in Tag 7F49h), then the DK_Identifier is an 8-byte zero padded slot_identifier based on the random slot_identifier generated by the owner device during key_sharing. If slot identifiers are provided online and BLE key material is provided by the vehicle OEM server (kBleOobKey and kBleIntroKey in Key Tracking Response Table 11-19), then DK Identifier is an 8-byte zero padded slot identifier based on the slot identifier sent in Key Tracking Response.
IV1	8	IV used by device for encrypting E1_Payload using Kble_intro
Tag1	4	AES-CCM authentication tag for E1_Payload using Kble_intro
E2_Payload	38	E2_Payload is containing BTAddrA (6 Byte) Ca (16 Byte) ra (16 Byte). The value of E2_Payload is encrypted (AES-128-CCM) using Kble_oob as shared secret. The BTAddrA, Ca and ra are Bluetooth address of the device at connection establishment,

Parameters	Length (bytes)	Description
		Calculated Value on the device, and Random Value on the device, respectively (see Figure 19-16).
IV2	8	IV used by device for encrypting E2_Payload using Kble_oob
Tag2	4	AES-CCM authentication tag for E2_Payload using Kble_oob

19.3.4.2 First Approach Response Message (FA-RS)

This message is sent by the vehicle to the device to share vehicle OOB data which is encrypted by Kble_oob key. The message and its parameters are defined in Table 19-56 and Table 19-57, respectively.

Table 19-56: First_Approach_RS message and its parameters.

Message	Message ID	Parameters
First_Approach_RS	0x0F	E_Payload IV, Tag

Table 19-57: Definition of the parameter for First_Approach_RS.

Parameters	Length (bytes)	Description
E_Payload	38	E_Payload is encrypted (AES-128-CCM) BTAddrB (6 Byte) Cb (16 Byte) rb (16 Byte) using Kble_oob as shared secret. The BTAddrB, Cb and rb, are the Bluetooth address of the vehicle at connection establishment, Calculated Value on the vehicle, and Random Value on the vehicle, respectively (see Figure 19-16).
IV	8	IV used by vehicle for encrypting vehicle OOB data using Kble_oob.
Tag	4	AES-CCM authentication tag for E_Payload using Kble_oob

19.3.5 Supplementary Service Message - RKE

19.3.5.1 RKE_Auth_RQ

This message is sent by vehicle upon receiving RKE action SubEvent (See Section [19.5.9](#)).

The Message Type and Message ID fields in RKE_Auth_RQ DK message (Table [19-19](#)) are set to Supplementary Service and RKE_Auth_RQ (14_h), respectively. The Data field in RKE_Auth_RQ DK message carries the TLV structure in Table 19-58. The Execution ID is set to the value received in the RKE Request SubEvent that has triggered the transmission of the RKE_Auth_RQ.

If the Selected_DK_Protocol_Version is 0300_h, then RKE_Auth_RQ DK message format is defined in this section in the Digital Key specification. If the Selected_DK_Protocol_Version is 0100_h, then RKE_Auth_RQ DK message format is defined in [\[41\]](#).

Table 19-58: TLV carried in RKE_Auth_RQ

Tag	Length (Bytes)	Description	Field is
95 _h	10 _h	RKE Challenge. This is the Vehicle unique challenge generated per	M

		RKE Request to avoid replay attack.	
84 _h	01 _h	Execution ID. It allows to uniquely identify the RKE_Auth_RQ associated with a RKE Request SubEvent.	M

19.3.5.2 RKE_Auth_RS

This message is sent by the device in response to the RKE_Auth_RQ.

The Message Type and Message ID fields in RKE_Auth_RS DK message (Table 19-19) are set to Supplementary Service and RKE_Auth_RS (15_h), respectively. The Data field in RKE_Auth_RS DK message carries the TLV structure in Table 19-59.

Arbitrary Data Attestation is described in Table 15-58. The arbitrary data (Tag 58_h) in Table 15-58 shall be set to SHA-256 hash of (RKE Challenge || Function ID || Action ID || Execution ID), wherein RKE Challenge and Execution ID values are provided in RKE_Auth_RQ. The device determines by itself the appropriate Function ID and Action ID values associated with the Execution ID indicated in the RKE_Auth_RQ. The usage field (Tag 93_h) in Table 15-58 is set D074DA4F_h or FC6F4C17_h.

If the Selected_DK_Protocol_Version is 0300_h, then RKE_Auth_RS DK message format is defined in this section in this Digital Key specification. If the Selected_DK_Protocol_Version is 0100_h, then RKE_Auth_RS DK message format is defined in [41].

Table 19-59: TLV carried in RKE_Auth_RS

Tag	Length (Bytes)	Description	Field is
7F2D _h	Variable	Arbitrary Data Attestation (All fields in Table 15-65).	M
84 _h	01 _h	Execution ID. Set to the value received in the corresponding RKE_Auth_RQ.	M

19.3.6 Supplementary Service Message – UWB Final Data

As part of the UWB ranging packet exchange described in Section 20.5.1, the device sends a Final_Data message to the vehicle containing the measured timestamps of all valid ranging responses. If the vehicle does not receive Final_Data, ranging is not successful.

The device must always send the Final_Data message over UWB. It may optionally also send message to the vehicle over Bluetooth LE. The delay to send the message over Bluetooth LE should be minimized to reduce latency on the vehicle side. Therefore, if supported by the device, the device shall send UWB Final data Supplementary Service Message for the most recently concluded ranging round within 3 connection events of completion of that ranging round.

The vehicle must always attempt to receive the Final_Data message over UWB. If the vehicle does not receive Final_Data over UWB, it may optionally wait for 3 connection events after conclusion of the ranging round (described in Section 20.5.1) to receive it over Bluetooth LE. The message and its parameters are defined in Table 19-60 and Table 19-61, respectively.

Table 19-60: UWB_Final_Data message and its parameters.

Message	Message ID	Parameters
UWB_Final Data	0x19	UWB_Session_ID Ranging_Block Encrypted_Final_Data

Table 19-61: Definition of the parameters for UWB_Final_Data.

Parameters	Length (bytes)	Value	Description
UWB_Session_ID	8	0 – 0xFFFFFFFF FFFFFFFF	ID of the UWB ranging session.
Ranging_Block	2	0 – 65535	Index of the ranging block
Encrypted_Final_Data	Variable	Variable	This parameter contains the encrypted Final_Data payload that is sent in the UWB ranging exchange described in Section 20.5.1. The unencrypted payload is described in Table 20-5 and Table 20-6, and the encryption algorithm is described Section 22.1.2. The vehicle shall use the UWB_Session_ID and Ranging_Block parameters to determine which key and nonce were used to encrypt the payload.

19.3.7 Vehicle OEM App Message

This message is used by the vehicle OEM's app or the vehicle to exchange messages with each other over the same L2CAP channel used in the Digital Key. Digital Key framework is agnostic of this message and its data structure. It is simply acting as pass-through. The message and its parameters are defined in Table 19-62 and Table 19-63, respectively.

Table 19-62: Pass_through message and its parameter.

Message	Message ID	Parameter
Pass_through	0x10	Payload

Table 19-63: Definition of the parameter for Pass_through.

Parameter	Length (bytes)	Description
Payload	Variable	Proprietary payload known to vehicle OEM's app and vehicle. Length of the payload depends on the device/vehicle payload size over Bluetooth LE limitations.

19.3.8 Head Unit Pairing Message

Head Unit Pairing messages enable head unit pairing between vehicle and device as part of the owner pairing as described in Section 19.5.1.

19.3.8.1 Head_Unit_Pairing_Preparation Message (HU-PP)

The message and its parameters are defined in Table 19-64 and Table 19-65, respectively.

Table 19-64: HU_PP message and its parameters.

Message	Message ID	Parameter
Head_Unit_Pairing_Preparation (HU-PP)	0x16	BT_Pairing_Configuration, BD_ADDR_Head_Unit, Vehicle capabilities

Table 19-65: Definition of the parameter for HU_PP.

Parameter	Length (bytes)	Description
BT_Pairing_Configuration	1	BitMask for supported capabilities: OOB_Configuration: Bit[0] : Value = 0. The vehicle head unit does not support OOB communication. Bit[0] : Value = 1. The vehicle head unit supports bidirectional OOB communication. Bit[1-7]: Reserved
BD_ADDR_Head_Unit	6	Bluetooth Device Address of the head unit as defined in Volume 2 Part H of [30]
Vehicle capabilities	2	BitMask for supported capabilities on vehicle Bit [1]: Value = 0: Head unit Bluetooth Classic (A2DP and HFP) pairing not supported Bit [1]: Value = 1: Head unit Bluetooth Classic (A2DP and HFP) pairing supported Bit [2]: Value = 0: Apple Carplay not supported Bit [2]: Value = 1: Apple Carplay Supported Bit [3]: Value = 0: Android Auto Not Supported Bit [3]: Value = 1: Android Auto Supported Bit [4]: Value = 0: Apple Wired Carplay not supported Bit [4]: Value = 1: Apple Wired Carplay supported Bit[5:15]: Reserved

19.3.8.2 Head Unit Pairing Request Message (HUP-RQ)

This message is sent by the device to the vehicle. The message and its parameters are defined in Table 19-66 and Table 19-67, respectively.

Table 19-66: HUP_RQ message and its parameters.

Message	Message ID	Parameter
Head_Unit_Pairing_RQ (HUP-RQ)	0x17	User Intent, BD_ADDR_Device, Device Capabilities, PairingDataR192, PairingDataR256, PairingDataC192, PairingDataC256

Table 19-67: Definition of the parameter for HUP_RQ.

Parameter	Length (bytes)	Description
User Intent	2	BitMask for user intent coming from device. Bit [0]: Value =0: Intent from device

Parameter	Length (bytes)	Description
		Bit [0]: Value =1: No intent from device Bit [1]: Value = 0: Head unit Bluetooth Classic usage not intended from user. Bit [1]: Value = 1: Head unit Bluetooth Classic usage intended from user Bit [2]: Value = 0: Apple Carplay usage not intended from user Bit [2]: Value = 1: Apple Carplay usage intended from user Bit [3]: Value = 0: Android Auto usage not intended from user Bit [3]: Value = 1: Android Auto usage intended from user Bit [4-15]: Reserved
BD_ADDR_Device	6	Bluetooth Device Address of the device as defined in Volume 2 Part H of [30].
Device Capabilities	2	BitMask for supported capabilities on device Bit [1]: Value = 0: Head unit Bluetooth Classic (A2DP and HFP) not supported Bit [1]: Value = 1: Head unit Bluetooth Classic (A2DP and HFP) supported Bit [2]: Value = 0: Apple Carplay not supported Bit [2]: Value = 1: Apple Carplay supported Bit [3]: Value = 0: Android Auto not supported Bit [3]: Value = 1: Android Auto supported Bit[4-15]: Reserved
PairingDataR192	16	Simple Pairing Randomizer R-192*
PairingDataR256	16	Simple Pairing Randomizer R-256*
PairingDataC192	16	Simple Pairing Hash C-192*
PairingDataC256	16	Simple Pairing Hash C-256*
* See volume 2 Part H Section 7.2.2 of [30]		

1 19.3.8.3 Head Unit Pairing Response Message (HUP-RS)

2 This message is sent by the vehicle to the device. The message and its parameters are defined in
 3 Table 19-68 and Table 19-69, respectively.

4 *Table 19-68: HUP_RS message and its parameters.*

Message	Message ID	Parameter
Head_Unit_Pairing_RS (HUP-RS)	0x18	BD_ADDR_Head_Unit, PairingDataR192, PairingDataR256, PairingDataC192, PairingDataC256

5 *Table 19-69: Definition of the parameter for HUP_RS.*

Parameter	Length (bytes)	Description
BD_ADDR_Head_Unit	6	Bluetooth Device Address of the head unit as defined in Volume 2 Part H of [30].
PairingDataR192	16	Simple Pairing Randomizer R-192 *
PairingDataR256	16	Simple Pairing Randomizer R-256 *
PairingDataC192	16	Simple Pairing Hash C-192 *
PairingDataC256	16	Simple Pairing Hash C-256 *

Parameter	Length (bytes)	Description
* See volume 2 Part H Section 7.2.2 of [30]		

19.3.9 DK Event Notification

DK Event Notifications are used to encapsulate events that are generated by the device or vehicle participating in the ranging exchange. The message and its parameters are defined in Table 19-70 and Table 19-71, respectively.

Table 19-70: DK Event Notification message and its parameters.

Message	Message ID	Parameter
DK Event Notification	0x11	Subevent_Category, Subevent_Code

Table 19-71: Definition of the parameters for DK Event Notification.

Parameter	Length (bytes)	Description
Subevent_Category	1	DK SubEvent category which are listed in the table below
Subevent_Code	Variable	SubEvent code as defined below per category.

19.3.9.1 DK SubEvent Category

Table 19-72 defines categories of DK SubEvents supported.

Table 19-72: DK SubEvents Category and its parameters.

DK SubEvent Category	SubEvent_Category	Parameter
Command Complete SubEvent	0x01	Command_Status
Ranging Session Status Changed SubEvent	0x02	Session_Status
Device Ranging Intent SubEvent	0x03	DR_Intent
Vehicle Status Change SubEvent	0x04	Vehicle_Status
RKE Request Subevent	0x05	Requested_Action
Head Unit Pairing SubEvent	0x06	Headunit_Pairing_Status
Device UA Policy SubEvent	0x07	UA Policy (see Table 19-86)
Device UA Status SubEvent	0x08	UA Status (see Table 19-87)
PDR Data Request SubEvent	0x09	PDR Data Request (See [45])
PDR Data SubEvent	0x0A	PDR Data (See [45])
RFU	0x0B - 0xFF	None

19.3.9.2 DK SubEvent Codes Per Category

Command Complete SubEvent

The Command Complete SubEvent is used to send appropriate code upon processing of last message. Table 19-73 and Table 19-74 detail the Command_Status codes applicable to the

‘Command Complete SubEvent’ category. For detailed flow on Command_Status SubEvents, see Section [19.7.1](#).

Table 19-73: Definition of Command_Status and its Command_Status codes.

Parameter	Length (bytes)	Value	Description
Command_Status	1	0x00 - 0xFF	See Table 19-74 for all command status codes

Table 19-74: List of Command_Status for Command Complete SubEvent.

Command_Status	Code	Description
Deselect_SE	0x00	Vehicle shall send this SubEvent to deselect the Digital Key applet. Upon receiving this, the device shall deselect the Digital Key applet. For each secure element transaction over Bluetooth LE (see SE message in Section 19.3.2), it shall be terminated by this command regardless of the transaction is successful or fail.
BLE_pairing_ready	0x01	Vehicle shall send this SubEvent code when its ready for Bluetooth LE pairing during Owner Pairing
Require_capability_exchange	0x02	Device shall send this SubEvent to signal a mismatched ranging capability or an update to the device capability. Upon receiving this, the Vehicle shall trigger ranging capability exchange (see Section 19.7.1.1)
Request_standard_transaction	0x03	During Owner Pairing and Friend First Approach, a device sends this SubEvent to signal the vehicle, when device has finished processing the preceding step in the Owner Pairing Flow or Friend First Approach. (See Section 19.5.8 for details)
Request_owner_pairing	0x04	Device shall send this SubEvent to request owner pairing
Reserved	0x05- 0x7F	Reserved for future use
General_error	0x80	Device or vehicle may send this SubEvent to signal an unknown error. Upon receiving this, the vehicle or device may retry or abort.
Device_SE_busy	0x81	Device may send this SubEvent to signal the SE is temporarily unavailable. The device may retry internally to get SE resources before sending this. Upon receiving this, the vehicle may retry the digital key flow.
Device_SE_transaction_state_lost	0x82	Device may send this SubEvent to signal standard transaction state was lost. Upon receiving this, the vehicle may retry the transaction
Device_busy	0x83	Device sends this SubEvent to signal device temporarily may not be able to respond to the sent request. Upon receiving this, the vehicle may retry the action.
Command_temporarily_blocked	0x84	Device shall send this SubEvent when the device UA state and UA policy do not allow the requested command.
Unsupported_channel_bitmask	0x85	Device shall send this SubEvent if the channels provided in the channel bitmask are currently not supported by the device
OP_Device_not_inside_vehicle	0x86	Vehicle shall send this SubEvent when the device was not found inside the vehicle during owner pairing. This error shall cause the vehicle and device to abort owner pairing
Device_resource_available	0x87	The device sends this subevent to signal to the vehicle that it recovered from a state where access to device internal resources was blocked. The vehicle may use this information to retry a flow that failed due to that previous device state.
Reserved	0x88- 0xFB	Reserved for future use

Command_Status	Code	Description
OOB_mismatch	0xFC	Device or vehicle may send this SubEvent to signal the incompatible OOB data exchanged during Bluetooth LE pairing. This is an error code for debugging purposes.
BLE_pairing_failed	0xFD	Device or vehicle may send this SubEvent to signal the failure in Bluetooth LE pairing. This is an error code for debugging purposes.
FA_crypto_operation_failed	0xFE	Device or vehicle may send this SubEvent to signal the failure in First approach due to cryptography. This is an error code for debugging purposes.
Wrong_parameters	0xFF	Device or vehicle may send this SubEvent to signal the use of unsupported message or format. This is an error code for debugging purposes.

1 Ranging Session Status Changed SubEvent

2 The Ranging Session Status Changed SubEvent is generated to indicate the start, progress and
3 completion of an UWB related action. These SubEvents can be generated by both the device and
4 the vehicle. The Session_Status codes are described in Table 19-75 and Table 19-76.

5 *Table 19-75: Definition of Session_Status and its Session_Status codes.*

Parameter	Length (bytes)	Value	Description
Session_Status	1	0x00 - 0xFF	See Table 19-76 for all session status codes.

6 *Table 19-76: List of Session_Status for Ranging Session Status Changed SubEvent.*

Session_Status	Code	Description
Ranging_session_URSK_refresh	0x00	Vehicle may send this SubEvent to request clean-up for pre-derived URSKs.
Ranging_session_URSK_not_found	0x01	Device shall send this SubEvent to signal, the device failed to find URSK for this UWB_Session_Id
Ranging_session_not_required	0x02	Device shall send this SubEvent when it detects the user does not intend to approach the vehicle e.g device is moving away from the vehicle. Upon receiving this subEvent, the vehicle may immediately suspend the ranging session.
Ranging_session_secure_ranging_failed	0x03	Vehicle shall send this SubEvent to signal the Vehicle failed to establish secure ranging.
Ranging_session_terminated	0x04	Vehicle shall send this SubEvent if it has stopped the ranging session and device shall send this SubEvent if it has stopped the ranging session (e.g. due to URSK TTL expiration).
Ranging_session_recovery_failed	0x06	Device shall send this SubEvent to signal it failed to recover ranging for this UWB_Session_Id
Ranging_session_suspended	0x07	Vehicle shall send this SubEvent only if it is suspending the current ranging session without allowing the responder to delay the suspension (e.g. UWB has temporarily become unavailable). Device shall send this SubEvent only if it is suspending the current ranging session without allowing the responder to delay the suspension (e.g. UWB has temporarily become unavailable). When vehicle or device receives this, it shall suspend its current ranging session as well without sending a Ranging_Suspend_RQ or a Ranging_session_suspended SubEvent.
Reserved	0x08-0xFF	Reserved for future

Device Ranging Intent SubEvent Codes

This set of SubEvents are generated by the device to notify the vehicle that the user may be approaching the vehicle and ranging may be required. The DR_Intent codes are described in [Table 19-77](#) and [Table 19-78](#).

Table 19-77: Definition of DR_Intent and DR_Intent codes.

Parameter	Length (bytes)	Value	Description
DR_Intent	1	0x00-0xFF	Low_approach_confidence, Medium_approach_confidence, High_approach_confidence, Reserved

Table 19-78: List of DR_Intent for Device Ranging Intent SubEvent.

DR_Intent	Code	Description
Low_approach_confidence	0x00	Device notifies vehicle of user approaching with low confidence
Medium_approach_confidence	0x01	Device notifies vehicle of user approaching with medium confidence
High_approach_confidence	0x02	Device notifies vehicle of user approaching with high confidence
Reserved	0x03- 0xFF	Reserved for future use

The device shall use the following definition of Confidence levels to select DR_Intent. The setting of the DR_Intent is up to the device OEM implementation. When the vehicle is in low power mode, the vehicle may decide to ignore ranging intent with low or medium approach confidence. When the vehicle is not in low power mode, and if the vehicle is ready to perform the secure ranging operation (e.g., no vehicle internal conflict), the vehicle shall initiate the secure ranging regardless of the DR_Intent confidence level.

To enable secure ranging based on a Device Ranging Intent, the device shall send a Device Ranging Intent SubEvent with at least low_approach_confidence between the point in time a Bluetooth LE connection with the vehicle is established and the point in time the vehicle triggers secure ranging (e.g. user touches the door handle).

The performance requirements from Section 19.2.3.3 shall apply when the vehicle starts the Ranging Session Setup flow after receiving a Device Ranging Intent SubEvent

Approach detection probability:

- Describes the probability an intent was sent in the last 2 to 20 seconds before the customer touched the door
- $P(\text{Approach detection}) = P(\text{in last [2s,20s] an intent was sent} \mid \text{Customer touches door handle})$

False approach detection probability:

- Describes the probability that a ranging intent was sent, but the customer has not arrived at the vehicle touching the door handle within the following 20s
- $P(\text{False approach detection}) = P(\text{Customer has not touched the door handle within the last 20s} \mid \text{ranging intent received 20s ago})$

Definition of Confidence levels:

- Low_approach_confidence:
 1. Approach detection probability: >90%, and
 2. False approach detection probability: as low as possible
- Medium_approach_confidence:
 3. Approach detection probability: as high as possible, and
 4. False approach detection probability: <75%
- High_approach_confidence
 5. Approach detection probability: as high as possible, and
 6. False approach detection probability: <50%

Vehicle Status Changed SubEvent

The Vehicle Status Changed SubEvent is used to inform the device about state changes of vehicle functions as shown in [Table 19-79](#). The Digital Key framework on the device shall store the latest values of received function state changes as long as the BLE connection is established. If the Selected_DK_Protocol_Version is 0300_h, then tag formats carried in the Vehicle Status Changed SubEvent are defined in Table 19-79. If the Selected_DK_Protocol_Version is 0100_h, then the tag formats carried in the Vehicle Status Changed SubEvent are defined in [41].

Table 19-79: List of Vehicle Status Changed SubEvent Tags.

The content of the Vehicle Status Changed SubEvent message is encoded as TLV format as defined in [1]				Length (bytes)	Description	Field is
7F71 _h				variable	Action execution status for function / action id. Tag is only present when an RKE or RKE-Hands-Free function was triggered beforehand or an automatic action was triggered due to the location / movement of the device (e.g., walk away locking).	C
	82 _h			1	Execution Status 00 _h : Function successfully executed 01 _h : Execution started 02 _h : Execution stopped successfully (upon user stop request) 03 _h : Execution ended (e.g., limit reached) 04 _h : Execution pending 05 _h –0F _h : Reserved 10 _h : Execution not possible or cancelled - unspecific reason 11 _h : Execution not possible due to vehicle state 12 _h : Authentication Error 13 _h : Device response timeout 14 _h : Device out of execution range (distance) 15 _h : Device out of execution range (angular) 16 _h : Execution not possible – maximum parallel executions reached 17 _h : Execution not possible – no active UWB ranging session	M

The content of the Vehicle Status Changed SubEvent message is encoded as TLV format as defined in [1]						Length (bytes)	Description	Field is
							18h–4Fh: Reserved 50h–EFh: Execution not possible or cancelled with OEM specific meaning. The meaning of these unsuccessful status codes is proprietary and may be used to send a more specific error reason to the vehicle OEM app. The definition is up to the vehicle OEM. F0h–FFh: Standardized error codes defined in Table 19-82.	
	84h					1	Execution ID. This tag is present when Execution ID value is known by the vehicle.	C
	87h					Variable	Arbitrary data with a maximum size of 64 bytes.	O
7F72h						variable	Vehicle function status summary. The tag is present, when the vehicle function status changes or the Actionable Now in User Flow property of an Action ID changes (see section 19.5.9.4).	C
	30h					variable	Sequence of vehicle function status	M
		A0h				variable	Vehicle function status	M
			80h			2	Function ID	M
			83h			1	Function status	M
			96h			1	Source of Change (see Table 19-80). It indicates the latest reason that caused the function status that is indicated in Tag 83h. See section 19.5.9.4 for conditions under which this tag is present in 7F72h.	M
			81h			Variable	Byte array of Action IDs that are actionable now from the current state. Each Action ID is 1 byte in length	M
			92h			variable	Byte array of Actionable Now in User Flow corresponding to each Action ID that is actionable now from the current state. Actionable now in User Flow is 1 byte in length. The bitmap value is below. Bit B0, set to 1 when the Action ID is actionable now in RKE. Bit B0, set to 0 when the Action ID is not actionable now in RKE. Bit B1, set to 1 when the Action ID is actionable now in RKE-Hands-Free with voice input modality. Bit B1, set to 0 when the Action ID is not actionable now in RKE-Hands-Free with voice input modality. Bit B2, set to 1 when Action ID is actionable now in RKE-Hands-Free with gesture input modality. Bit B2, set to 0 when Action ID is not actionable now in RKE-Hands-Free with gesture input modality. Bits B3 - B7: RFU. These are set to zero by vehicle and ignored by the device in this version of the specification.	M
			87h			variable	Arbitrary data with a maximum size of 64 bytes.	O
7F75h						variable	Request confirmation for enduring RKE action. Only present, when enduring RKE action is in progress.	C
	84h					1	Execution ID	M

The content of the Vehicle Status Changed SubEvent message is encoded as TLV format as defined in [1]						Length (bytes)	Description	Field is
	85 _h					1	Execution Percentage integer value between 00 _h and 64 _h . Values 65 _h - FF _h are RFU.	M
	87 _h					variable	Arbitrary data with a maximum size of 64 bytes.	O
7F78 _h						3	Device Ranging Intent SubEvent notification feedback. Vehicle sends this tag to the device indicating region in which the device is present when Device Ranging Intent SubEvent notification is received and after localizing the device with UWB ranging (see section 19.5.9.5).	C
	94 _h					1	Feedback Qualifier. 00 _h : Device is outside the welcome zone of the vehicle. 01 _h : Device is inside the welcome zone but outside the passive entry zone. 02 _h : Device is inside the welcome zone and inside the passive entry zone. 03 _h -FF _h : RFU Welcome zone and passive entry zone are two zones around the vehicle. In welcome zone convenience features such as 'welcome lights' are available. In passive entry zone, (un)locking features are available.	M
7F79 _h							Supported Function and Action summary	C
	30 _h					variable	Sequence of vehicle functions descriptor	M
		A0 _h				variable	Vehicle function descriptor	M
			80 _h			2	Function ID	M
			81 _h			variable	Byte array of Action IDs that are supported. Each Action ID is 1 byte in length	M
			89 _h			variable	Byte array of Supported in User Flow corresponding to each Action ID that is supported. Supported in User Flow is 1 byte in length. The bitmap value is below. Bit B0, set to 1 when the Action ID is supported in RKE. Bit B0, set to 0 when the Action ID is not supported in RKE. Bit B1, set to 1 when the Action ID is supported in RKE-Hands-Free with voice input modality. Bit B1, set to 0 when the Action ID is not supported in RKE-Hands-Free with voice input modality. Bit B2, set to 1 when Action ID is supported in RKE-Hands-Free with gesture input modality. Bit B2, set to 0 when Action ID is not supported in RKE-Hands-Free with gesture input modality. Bits B3 - B7: RFU. These are set to zero by vehicle and ignored by the device in this version of the specification.	M
			93 _h			variable	Byte array of Function Qualifier corresponding to each Action ID that is supported. Function Qualifier is 1 byte in length. The bitmap value is below.	

The content of the Vehicle Status Changed SubEvent message is encoded as TLV format as defined in [1]						Length (bytes)	Description	Field is
							Function Qualifier defines regions around the vehicle where the Action ID is supported with RKE-Hands-Free with input modality voice or gesture. Function Qualifier does not apply to RKE actions. Set the bit to 0 if the Action is not supported in a region. Set the bit to 1 if the Action is supported in a region. B0: Driver side B1: Passenger side B2: In Front of vehicle B3: Behind rear of vehicle B4: Inside RKE-Hands-Free distance (coverage circle around the vehicle with RKE-Hands-Free distance as radius) B5: Outside RKE-Hands-Free distance B6-B7: RFU. RFU shall be set to 0 and ignored on reception in this Digital Key specification. See section 19.5.9.4 for conditions under which this tag is present in 7F79_h.	
5F78 _h						1	Subscription status changed, only present, when the vehicle changes the subscription state 0: Unsubscribed all, no subscription possible 1: Subscription request successful 2: Unsubscribe request successful 3: Subscription possible/resubscription required	C

Tag 7F71_h shall always be sent together with 7F72_h in a single Vehicle Status Changed SubEvent message to the device that triggered the action causing the vehicle state to change. An exception is when the vehicle state takes more than 250ms to change after the action received. In this situation, 7F71_h and 7F72_h can be sent separately in two different Vehicle Status Changed SubEvent messages.

If the vehicle takes more than 3 seconds to execute after the action is received, the vehicle shall send 7F71_h with execution status set to ‘execution pending’ for the corresponding Function ID. The vehicle should send tag 7F71_h with the execution status set to ‘execution pending’, every 2 seconds until requested enduring action is started or event action is executed, or an enduring/event action is cancelled on the vehicle or device side. The vehicle can terminate an action if its execution is pending longer than a timeout duration. The value of timeout duration is determined by vehicle. The device can stop an enduring/event action whose execution has not started by sending 7F77_h in RKE Request SubEvent.

When sent together, the templates order shall be always 7F71_h followed by 7F72_h. In case of automatic actions triggered due to the location of the device, the device that triggered a state change will receive 7F71_h and 7F72_h as a single SubEvent message. Other BLE connected

devices (not used in decision making) will only receive the 7F72_h template in the SubEvent notification.

The values of Source of Change (tag 96_h) are listed in the Table 19-80. If the vehicle cannot determine which connected device triggered an action, it shall report the following Source of Change values to all connected devices:

- 02_h: in case of RKE action operation
- 06_h: in case of RKE-Hands-Free operation
- 04_h: in case of passive entry operation

Table 19-80: Source of Change values

Source of Change Value	Description
00 _h	Operation that caused the function status to change came from keyfob.
01 _h	Operation that caused the function status to change came from inside the car (e.g., head unit, steering wheel buttons, etc.)
02 _h	Operation (RKE action) that caused the function status to change came from the device receiving this this Tag 7F72 _h
03 _h	Operation (RKE action) that caused the function status to change did not come from the device receiving this Tag 7F72 _h
04 _h	Operation (Passive Entry) that caused the function status to change came from the device receiving this this Tag 7F72 _h
05 _h	Operation (Passive Entry) that caused the function status to change did not come from the device receiving this this Tag 7F72 _h
06 _h	Operation (RKE-Hands-Free) that caused the function status to change came from the device receiving this this Tag 7F72 _h
07 _h	Operation (RKE-Hands-Free) that caused the function status to change did not come from the device receiving this this Tag 7F72 _h
08 _h	Operation that caused the function status to change came from a trigger from the exterior of the vehicle (e.g., door handle, trunk button, frunk button, etc.)
09 _h	Operation that caused the function status to change came from the car (e.g., car in motion, automatic lock, etc.)
0A _h	Unknown
0B _h – FF _h	RFU

[Table 19-81](#) lists the Function ID and its corresponding Action IDs and/or status values. Depending on the usage of the Function ID, it may not provide the corresponding Action ID or status value. For certain actions that represent automatically executable functions, e.g., due to device position or movement, the vehicle may send an execution status without an explicit RKE trigger. An example of this is using it to signal when execution of an automatic action is unsuccessful (e.g., walk away lock with vehicle state not allowing to lock the vehicle).

1

Table 19-81: Definition of Function ids and their corresponding Action ids.

Function id	Function	Possible action ids	Possible function status values (Note 1)	Event-based or enduring action	Device (Note 2)	Vehicle (Note 2)
0000_h-000F_h Central locking category (vehicle may support up to one of the following options)						
0003 _h	Universal Lock/Unlock	01 _h : Lock and No Arm 02 _h : Lock and Partial Arm 03 _h : Lock and Full Arm 04 _h -0A _h : RFU (Lock actions) 0B _h : Driver Door unlock 0C _h : Unlock 0D _h : Unlock All 0E _h -4F _h : RFU 50 _h -FF _h : RFU The Front door refers to both left and right front door. Rear door refers to both left and right rear doors.	01 _h : Locked and Not Armed 02 _h : Locked and Partial Armed 03 _h : Locked and Fully Armed 04 _h -0A _h : RFU (Lock status) 0B _h : Driver Door Unlocked 0C _h : Unlocked 0D _h : All Unlocked	Actions 01 _h , 02 _h , 03 _h , 0B _h , 0C _h and 0D _h are Event-based.	M	M
0004 _h ... 000F _h	Reserved for standardization					
0010 _h	Driving Readiness		0: No Driving Readiness 1: Driving Readiness (e.g., engine “running” for ICE vehicles)	Event	-	M
0011 _h	Vehicle low power Mode		0: Normal power mode 1: Low power mode	Event	-	O
0012 _h	Low Fuel Status		0: Fuel not low 1: Fuel Low	-	-	O
0013 _h ... 001F _h	Reserved for standardization					
0100 _h	Remote Climatization	0: Stop Climatization 1: Start Climatization	0: Remote Climatization not active 1: Remote Climatization active	Event	O	O
0101 _h	Panic alarm	0: Mute panic alarm 1: Trigger panic alarm	0: Panic alarm not active 1: Panic alarm active	Event	M	O
0102 _h	Fuel Lid	1: Release	0: Closed 1: Open	Event	O	O
0103 _h	Charging Lid	1: Release	0: Closed 1: Open	Event	O	O
0104 _h	Powered Charging Lid	0: Close 1: Open	0: Closed 1: Open	Open: enduring without confirmation Close: enduring	O	O

Function id	Function	Possible action ids	Possible function status values (Note 1)	Event-based or enduring action	Device (Note 2)	Vehicle (Note 2)
				without confirmation		
0105 _h	Charging Port Commands	0: Charging Port unlock 1: Charging Port Lock	0: Charging Port unlock 1: Charging Port Lock	Event	O	O
0106 _h	Charging	0: Charging stop 1: Charging start	0: not charging 1: charging	Event	O	O
0107 _h	Remaining Battery range		Percentage integer value between 00 _h and 64 _h	Event	O	O
0108 _h ... 010F _h	Reserved for standardization					
0110 _h -011F _h Rear Vehicle Storage Variants category (the vehicle should support one of the following options if the vehicle has a trunk)						
0110 _h	Manual Trunk/Decklid/ Liftgate/ Tailgate Controls	1: Release	0: Closed 1: Open	Event	O	O
0111 _h	Power Trunk/Decklid/ Liftgate/ Tailgate Controls (with confirmation for close)	0: Close 1: Open	0: Closed 1: Open	Open: enduring without confirmation, Close: enduring with confirmation	O	O
0112 _h	Power Trunk/Decklid/ Liftgate/ Tailgate Controls (without confirmation for close)	0: Close 1: Open	0: Closed 1: Open	enduring without confirmation	O	O
0113 _h ... 011F _h	Reserved for standardization					
0120 _h -012F _h Front Vehicle Storage Variants category (the vehicle should support one of the following options if the vehicle has a frunk)						
0120 _h	Manual “Frunk” Front Trunk Controls Traditional vehicle body style	1: Release	0: Closed 1: Open	Event	O	O
0121 _h	Power “Frunk” Front Trunk Controls with confirmation	0: Close 1: Open	0: Closed 1: Open	Open: enduring without confirmation,	O	O

Function id	Function	Possible action ids	Possible function status values (Note 1)	Event-based or enduring action	Device (Note 2)	Vehicle (Note 2)
	Traditional vehicle body style			Close: enduring with confirmation		
0122 _h	Power “Frunk” Front Trunk Controls without confirmation Traditional vehicle body style	0: Close 1: Open	0: Closed 1: Open	Open/Close: enduring without confirmation	O	O
0123 _h ... 012F _h	Reserved for standardization					
0130 _h -013F _h Vehicle Power Doors Variants category (if supported by vehicle)						
0130 _h	Power Front Left with Confirmation	0: Close 1: Open	0: Closed 1: Open	enduring with confirmation	O	O
0131 _h	Power Front Left without confirmation	0: Close 1: Open	0: Closed 1: Open	enduring without confirmation	O	O
0132 _h	Power Front Right with Confirmation	0: Close 1: Open	0: Closed 1: Open	enduring with confirmation	O	O
0133 _h	Power Front Right without confirmation	0: Close 1: Open	0: Closed 1: Open	enduring without confirmation	O	O
0134 _h	Power Rear Left with Confirmation	0: Close 1: Open	0: Closed 1: Open	enduring with confirmation	O	O
0135 _h	Power Rear Left without confirmation	0: Close 1: Open	0: Closed 1: Open	enduring without confirmation	O	O
0136 _h	Power Rear Right with Confirmation	0: Close 1: Open	0: Closed 1: Open	enduring with confirmation	O	O
0137 _h	Power Rear Right without confirmation	0: Close 1: Open	0: Closed 1: Open	enduring without confirmation	O	O
0138 _h	Front Door left contact status		0: Closed 1: Open	Event	O	O
0139 _h	Front Door right contact status		0: Closed 1: Open	Event	O	O
013A _h	Rear Door left contact status		0: Closed 1: Open	Event	O	O
013B _h	Rear Door right contact status		0: Closed	Event	O	O

Function id	Function	Possible action ids	Possible function status values (Note 1)	Event-based or enduring action	Device (Note 2)	Vehicle (Note 2)
			1: Open			
013C _h ... 013F _h	Reserved for standardization					
0140 _h -0141 _h Vehicle Power Windows / Roof (if supported by vehicle)						
0140 _h	Power Windows	0: Close 1: Open	0: Closed 1: Open	enduring with confirmation	O	O
0141 _h	Power roof	0: Close 1: Open	0: Closed 1: Open	enduring with confirmation	O	O
0150 _h -015F _h Vehicle Configuration category						
0150 _h	Sharing disablement	0: RFU 1: Disable sharing for all keys (OPT OUT)	0: Sharing enabled for all keys (OPTED IN) 1: Sharing disabled for all keys (OPTED OUT)	Event	O	O
0151 _h ... 05FF _h	Reserved for standardization					
0600 _h to FFFF _h	Vehicle OEM-proprietary functions					
Note 1: Possible function status values: Range from 00 _h to EF _h , for values F0 _h to FF _h (see Table 19-82) Note 2: Device and Vehicle codes: Mandatory, Optional, Conditional						

- 1 Generic function status values shall be used to indicate errors or the absence of a certain
- 2 function. These function status values are defined in Table 19-82.

- 3 *Table 19-82: Definition of Standardized Function/Execution Status Values to Indicate the (Temporary)*
- 4 *Unavailability of Function/Execution or Errors.*

Function status value	Possible function status values
F0 _h	Function not supported by vehicle
F1 _h	Action not supported by vehicle
F2 _h	Function temporarily not available
F3 _h	Action temporarily not available
F4 _h	Function not purchased
F5 _h	Action not purchased
F6 _h	Insufficient entitlements
F7 _h	Use of wrong RKE security policy for requested action
F8 _h to FD _h	Reserved
FE _h	Generic error, no further description available
FF _h	Reserved, shall be used if action is supported but function has no explicit status

RKE Request SubEvent

The RKE Request SubEvent's content, described in Table 19-83, is encoded as TLV. The RKE Request SubEvent may be sent:

1. in response to user's request to perform one of the defined RKE or RKE-Hands-Free features (tag 7F70_h and tags 7F76_h for enduring actions).
2. to subscribe to vehicle function status updates (tag 7F73_h).
3. to get the current vehicle functions' status (tag 7F74_h).
4. in response to user's request to stop an enduring RKE or RKE-Hands-Free action or in response to a stopped enduring RKE or RKE-Hands-Free action upon receiving the 7F75_h confirmation request (tag 7F77_h).
5. in response to user's request to stop an enduring/event RKE or RKE-Hands-Free action whose execution has not started (tag 7F77_h).

The Execution ID field in RKE Request SubEvent allows to identify parallel ongoing RKE function executions.

If the Selected_DK_Protocol_Version is 0300_h, then tag formats carried in RKE Request SubEvent are defined in Table 19-83. If the Selected_DK_Protocol_Version is 0100_h, then tag formats carried in RKE Request SubEvent are defined in [41].

Table 19-83: Definition of RKE Request SubEvent Templates.

Tag			Length (bytes)	Description	Field Is
7F70 _h			Variable	Request RKE or RKE-Hands-Free action. Only present, when RKE action shall be triggered.	C
	80 _h		2	Function ID	M
	81 _h		1	Action ID	M
	84 _h		1	Execution ID. It is set randomly.	M
	87 _h		Variable	Arbitrary data with a maximum size of 64 bytes.	O
	88 _h		1	Action Type. 00_h: RKE 01_h: RKE-Hands-Free with voice input modality 02_h: RKE-Hands-Free with gesture input modality 03_h-FF_h: RFU	M
	90 _h		2	Device OEM Identifier. This tag is present if the Action Type is RKE-Hands-Free	C
7F73 _h			variable	Subscribe to vehicle function status events. Unsubscribing from all status updates is possible by setting "from function id" and "to function id" to 0. Tag is only present, when subscription change shall be performed.	C
	30 _h		variable	Sequence of function ids. Only present, if vehicle has subscription change to the list of function status	C
		84 _h	2	From function id	M
		85 _h	2	To function id	M
	86 _h		0	If present, vehicle shall send an immediate update of supported function states in this range.	C
7F74 _h			variable	Get status update for function ID. Only present, when explicit status updates shall be requested.	C
	30 _h		variable	Sequence of function IDs. Only present, if status for a list of function ids shall be requested.	C
		80 _h	variable	Function ID	M

Tag		Length (bytes)	Description	Field Is
	86 _h	0	If present, vehicle shall send a full status update of all supported functions and their corresponding states.	C
7F76 _h		variable	Continue enduring RKE or enduring RKE-Hands-Free action. Only present, when enduring RKE or enduring RKE-Hands-Free action is in progress.	C
	84 _h	1	Execution ID. It is set to the value of Execution ID indicated in the immediately preceding Tag 7F75 _h for this Function ID.	M
	87 _h	Variable	Arbitrary data with a maximum size of 64 bytes.	O
	88 _h	variable	Arbitrary data (max. 64 byte)	O
7F77 _h		3	Stop enduring RKE or enduring RKE-Hands-Free action. Only present, when the enduring action in progress shall be stopped or any enduring/event RKE/RKE-Hands-Free action, whose execution is not started is being stopped.	C
	84 _h	1	Execution ID. It is set to the value of Execution ID indicated in the immediately preceding Tag 7F75 _h or 7F70 _h for this Function ID.	M

Head Unit SubEvents

The Head Unit SubEvent is generated to indicate the final success or failure of the transaction or to signal application-specific codes to the device. The Head Unit SubEvent codes are described in Table 19-84 and Table 19-85.

Table 19-84: Definition of Headunit_Pairing_Status and its Session_Status.

Parameter	Length (bytes)	Value	Description
Headunit_Pairing_Status	1	0x00 - 0xFF	See Table 19-85 for all head unit pairing status codes

Table 19-85: List of Session_Status for Head Unit SubEvent.

Session_Status	Code	Description
Headunit_pairing_success	0x00	Head Unit Pairing is successful
Headunit_pairing_fail	0x01	Head Unit Pairing fails
Reserved	0x02-0xFF	Reserved for future

Device UA Policy SubEvent

The Device UA Policy SubEvent is used by Device to inform Vehicle about the current used UA Policy on the Device. When the key is created, the UA Policy is set to Off [O] by default. At BLE connection setup Device should inform Vehicle if the UA Policy has changed to other value than Off [O]. During BLE connection, Device should also inform Vehicle if UA Policy has changed.

Table 19-86: List of UA Policies

UA Policy	Value	Description
Off [O]	0x00	UA never requested (default)
Access [A]	0x01	UA requested on first access or first engine start
Access [AL]	0x02	UA requested on first access and lock or first engine start and lock
Engine [E]	0x03	UA on every engine start

UA Policy	Value	Description
Always [T]	0x04	UA on every usage
others	0x05 – 0xFF	RFU

Device UA Status SubEvent

If UA Policy is Off [O], Device UA Status SubEvent should not be sent. If it's sent then it should be ignored by Vehicle.

If UA Policy is not Off [O], Device UA Status SubEvent is used by Device to inform Vehicle about the current UA Status on the Device. The UA Status is ON by default.

Table 19-87: List of UA Policies

UA Status	Value	Description
OFF	0x00	Device doesn't require UA for a requested DK function due to: - Device is in a state where only DK function can be triggered (see section 9.1) or - grace period is active during implicit UA (see section 9.2)
ON	0x01	Device might require UA for a requested DK function (default).
others	0x02 – 0xFF	RFU

PDR Data Request SubEvent

This SubEvent is sent by Vehicle to request Device to provide PDR Data. PDR data can be used by the vehicle in conjunction with the UWB ranging data or other localization methods to calculate the relative location of the device. The parameters of the SubEvent are described in [Table 19-88](#)

Table 19-88: PDR Data Request Parameters

Parameter	Length (bytes)	Description
Request Flag	1	Flag used to request Device to start, stop, update PDR data. 0x00: Vehicle requests Device to stop providing PDR data. 0x01: Vehicle requests Device to start providing PDR data. Vehicle may also use this value to request Device to provide PDR data with updated frequency. 0x02 – 0xFF: RFU
Frequency	1	Frequency for Device to provide PDR data. Only applicable if Request Flag is set to 0x01. 0x00: 5 Hz 0x01: 10 Hz 0x02 – 0xFF: RFU

PDR Data SubEvent

If requested by Vehicle (see Request Flag in [Table 19-88](#)), this SubEvent is periodically sent by Device to Vehicle to provide PDR data. The frequency to send is defined in [Table 19-88](#). The parameters of the SubEvent are described in [Table 19-89](#).

Table 19-89: PDR Data Parameters

Parameter	Length (bytes)	Description
Pdr_x [mm]	2	Data type: integer
Pdr_y [mm]	2	Data type: integer
Pdr_z [mm]	2	Data type: integer

19.4 Time Synchronization

The time synchronization shall be mandatory for the device. The time synchronization support is strongly recommended for the vehicle due to performance gains such as improved latency and power saving benefit for the vehicle.

19.4.1 Definitions

The following definitions are used for this subsection:

19.4.1.1 Device UWB Clock and UWB Device Time

Device UWB Clock is defined as the UWB clock of the device as it appears to the vehicle.

UWB Device Time is defined as the timestamp on the Device UWB Clock.

The appearance of the Device UWB Clock to a vehicle can be defined or undefined:

1. “Defined”

- Device UWB Clock becomes “defined” to the vehicle, once a successful Bluetooth LE Timesync procedure with valid UWB_Device_Time has been performed
- Device UWB Clock becomes “defined” to the vehicle, once an UWB packet (of a ranging session with this device) has been received
- Once defined, vehicle expects that UWB device clock is operated in a consistent manner and the characteristics and requirements below hold

2. “Undefined”

- Device UWB Clock appears “undefined” to the vehicle prior to a successful Bluetooth LE Timesync.
- Note: This is the case if the Device UWB clock has not been started.
- Device UWB Clock becomes “undefined”, if there is no UWB and no Bluetooth LE connection for more than 30 seconds.
- Device UWB Clock becomes “undefined”, if a ranging session is suspended.
- Device UWB Clock becomes “undefined”, if a ranging session is terminated.
- Note: Session is terminated if 12 hours lifetime is exceeded or the STS-Index range is exhausted.

Characteristics of Device UWB Clock

- UWB clock is UWB ranging session specific.

Note: Device may use a common clock for all its ranging sessions and maintain this clock even if a session is suspended.

2. The Device UWB Clock may have any value when entering “defined” state (within its specified range). For example, the device may set UWB_Device_Time=0 upon first valid Bluetooth LE Timesync procedure.
3. During an active ranging session, the Device UWB Clock is identical to the initiator clock as specified in Section [20.2](#).
4. Suspending a ranging session shall put the UWB clock into an undefined state and reset it to an arbitrary value for session recovery.

Note: UWB_Time0 for resuming a ranging session may be smaller than UWB_Device_Time at ranging session suspend.

5. Device UWB clock should have a granularity of 1 μ sec or smaller, considering that the format in the timesync message and the RSS-RS / RR-RS/ CRR-RS message is defined with 1 μ sec resolution.
6. The range of the UWB clock is limited to $0 \dots 2^{64}-1$ μ sec due to the format in the timesync message and the RSS-RS / RR-RS/ CRR-RS message.
7. Device UWB clock does not depend on the existence of a BT connection as long the ranging session is not suspended.

Note: A short Bluetooth reconnection that does not trigger a ranging suspension during an active ranging session shall trigger Bluetooth LE Timesync procedure 0. But the short Bluetooth reconnection shall not impact the operational state of Device UWB clock (while in “defined” mode).

Requirements for Device

1. Device shall be able to map a timing reference (see Section [19.3.3](#)) onto the Device UWB Clock and generate a UWB_Device_Time timestamp. The term “to map” here means, to take a timestamp from the Device UWB Clock in the moment of the Bluetooth LE Timesync defined timing reference on the air interface, or equivalently, compute the value that was held by the Device UWB Clock during the timing reference by means of an OEM specific method.
2. Device shall support at least one successful Bluetooth LE Timesync procedure before session start/resume.
3. Device shall support all vehicle triggered timesync procedures (Bluetooth LE Timesync procedure 1) during an active session.
4. All (valid) UWB_Device_Time values provided in the timesync messages shall be consistent clock states in accordance with the MAC grid as specified in Section [20.2](#).
 - Note:* In particular, the value of the UWB clock at session start shall be UWB_Time0 (as announced in the RSS-RS/RR-RS message).
5. UWB_Device_Time shall be presented as the clock state of a 64-bit counter with 1 μ sec resolution; the 64-bit counter shall wrap around at $2^{64}-1$.

Tolerance of Device UWB Clock frequency:

1. Before session start (or before session resume):
 - Maximum tolerance of UWB clock frequency is Device_max_PPM, as included in timesync message
2. After session start (during active ranging session):
 - Maximum tolerance of UWB clock frequency is Device_max_PPM, as included in timesync message and shall be less than ± 100 ppm as specified in Section 6.9.7.2 of [33].

19.4.1.2 Terminology: Clock Offset, Skew, Drift

The terminology used in this section follows Section 10 of [37].

Clock state: Value of a clock x at time $t=T$

Clock offset: Clock offset is the difference in clock states between a clock x and the true time at an arbitrary point in time T .

Relative clock offset: Relative clock offset is the difference in clock states between a clock x and a clock y , at an arbitrary point in time T .

Clock skew: Clock skew is the difference in clock frequency to the nominal clock frequency.

- Clock skew can be viewed as first derivative in time of clock offset.
- Clock skew normalized to the nominal frequency and multiplied by 10^6 is “clock skew in ppm”.

Relative clock skew: Relative clock skew is the difference in clock frequencies of clock x and clock y .

- Relative clock skew can be viewed as the first derivative in time of relative clock offset (the difference in clock frequencies at certain point in time).

Clock drift: Clock drift is the derivative of clock skew (the variation of clock frequency over time). Clock drift is assumed to be negligible for the purpose of this specification.

Figures of merit for timesync

- Vehicle UWB Clock (the UWB clock in a vehicle anchor = responder) is clock x .
- Device UWB Clock is the clock y , and is the reference clock that defines the UWB MAC grid, and to which the responder needs to synchronize.
- Timesync procedure is used to determine the relative clock offset between Device UWB clock and Vehicle UWB clock for the time of the BLE event on the air interface
- The worst-case clock skew Device_max_PPM of the device UWB clock (relative to true time) and the worst-case clock skew of the vehicle can be used to determine an upper bound for the relative clock offset of a future event (like session start).
- Vehicle may estimate the relative clock skew and compensate for it, to minimize the relative clock offset of a future event (like session start); for the time interval of interest the clock drift is assumed to be zero.

19.4.2 General Description

The Bluetooth connection serves as a common time domain between the vehicle and device and is used to synchronize the Bluetooth-connected UWB transceiver by mapping a shared BT event into the UWB clock domain. Synchronization methods are shown in Figure 19-6.

UWB Packet reception:

The reception of an UWB packet by the vehicle should allow to perform a better synchronization than the Bluetooth LE Timesync synchronization method described in this chapter.

Because this gives an anchor the precise position within the MAC grid (see Section 20.3), which it may be used to adjust the relative clock offset between device and anchor. This subsection describes the “Bluetooth LE Timesync” synchronization which happens before the reception of an UWB packet by the vehicle.

Note: An anchor may set its UWB-RX into scanning mode and wait for the reception of a packet without using any “Out of band” synchronization. This is always possible, at the cost of increased power consumption at the vehicle and other potential constraints.

Bluetooth LE Timesync:

In the absence of UWB Packet reception, this section describes a time synchronization over Bluetooth.

Time synchronization over Bluetooth

Time synchronization over Bluetooth, referred as “Bluetooth LE Timesync”, is an out-of-band synchronization method that is carried out between the device and one particular anchor. There are two types of “Bluetooth LE Timesync” procedure (see to Section 19.4.4):

- “Bluetooth LE Timesync at Bluetooth connection”, also referred as “Procedure 0”
- “Bluetooth LE Timesync triggered by vehicle”, also referred as “Procedure 1”

References in this chapter to “Bluetooth LE Timesync” apply to both “Bluetooth LE Timesync at Bluetooth connection” and “Bluetooth LE Timesync triggered by vehicle”

Vehicles may use “Bluetooth LE Timesync” to synchronize on the start of the UWB ranging session.

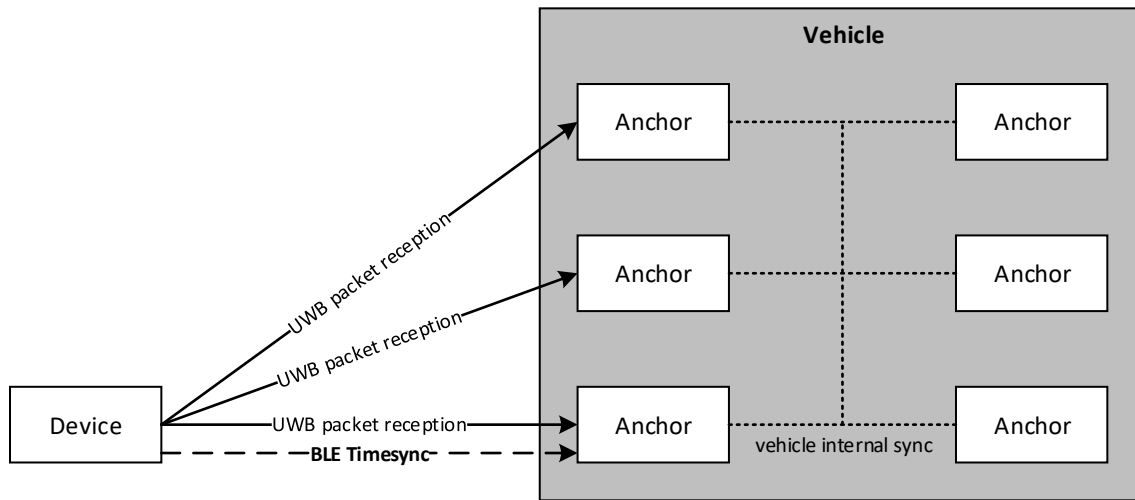
After start of the session the vehicle may use “Bluetooth LE Timesync” to re-synchronize on the MAC grid if UWB connection was lost for a longer period.

Vehicle internal synchronization

Vehicle internal synchronization may be used to distribute device synchronization (achieved by a particular anchor based on UWB packet reception or “Bluetooth LE Timesync”) between the anchors. The mechanism to achieve the vehicle internal synchronization is implementation specific and is out of scope for this specification.

The different synchronization methods are shown in Figure 19-6.

Figure 19-6: Synchronization Methods.

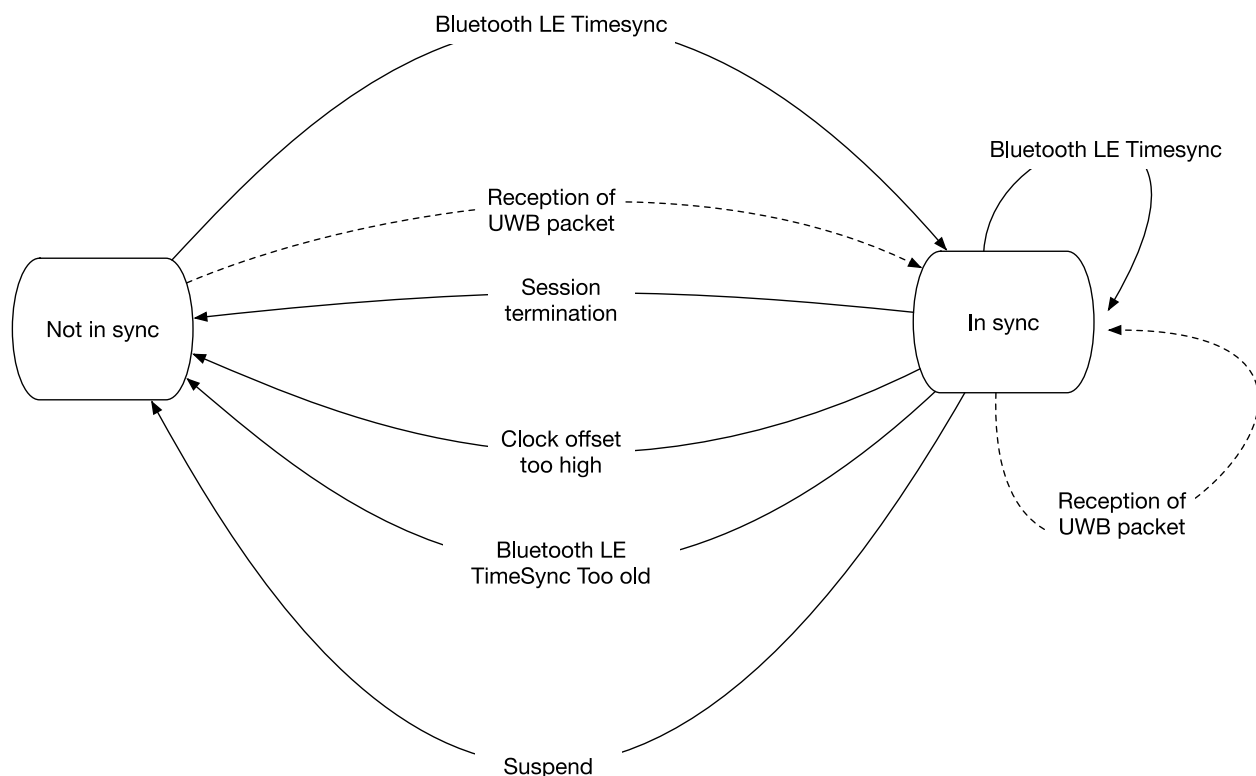


There are two states for the time synchronization between device and anchor as shown in Figure 19-7. Those states are only known by the vehicle.

- “not in sync”: Clock offset of anchor UWB clock vs device UWB clock is unknown, or potential offset is larger than 1ms
- “in sync”: Clock offset of anchor UWB clock vs device UWB clock is better than 1ms

Note: The limit for the relative clock accuracy that defines the “in sync” state is up to the vehicle implementation.

Figure 19-7: Synchronization state between device and anchor.



Note: This state diagram defines the pre-conditions for time synchronization and assumes a vehicle implementation which establishes this time synchronization in all anchors. In a real implementation there could be further boundary conditions, that might hinder time synchronization, although the conditions from this state diagram are given.

Note: Reception of an UWB packet shall lead immediately to the “In sync” state. This synchronization via reception of an UWB packet is indicated as dashed line in the state diagram is referred as “In-band” UWB synchronization. However, the scope of this Section is “Out-of-band” UWB synchronization via Bluetooth LE Timesync.

If “in sync”, the vehicle shall be able to predict events like session start or a position within the MAC grid (for an active ranging session) with some uncertainty.

The uncertainties are caused by clock skew between device and vehicle UWB clock, UWB_Device_Time uncertainty in case of “Bluetooth LE Timesync”, and uncertainties on vehicle side.

If no ranging session was started or recovered within 10 seconds after a successful Bluetooth LE Timesync, the Device UWB Clock becomes “Not in sync”.

The vehicle shall only trigger “Bluetooth LE Timesync” if conditions in condition set A or condition set B is met:

Condition Set A:

1. Condition A.1: A ranging session was needed in the last 150s or is currently running. This can be signaled by Device Ranging Intent as sent by device to vehicle, user physical intent on vehicle (e.g. touching door handle, engine start), or when Session Ranging Setup is to be performed (e.g. during owner pairing, First New Key Transaction, passive entry).
2. Condition A.2: Previous successful “Bluetooth LE TimeSync” was done (either by device or vehicle) more than 150s ago.

Condition Set B:

1. Condition B.1: A ranging session is needed. This can be signaled by Device Ranging Intent as sent by device to vehicle, user physical intent on vehicle (e.g. touching door handle, engine start), or when Session Ranging Setup is to be performed (e.g. during owner pairing, First New Key Transaction, passive entry).
2. Condition B.2: No active ranging session is on-going. None of the vehicle’s UWB anchors produced a valid ToF measurement within 10 seconds since the last “Bluetooth LE TimeSync”
3. Condition B.3: Previous successful “Bluetooth LE TimeSync” was done (either by device or vehicle) more than 10s ago.

Device clock skew is upper bounded by the parameter Device_max_PPM.

The vehicle may estimate the relative UWB clock skew (UWB clock frequency offset between vehicle and device) using the information from the most recent and one or more previous

timesync procedures. In this case, the vehicle may use the estimated clock skew and compensate for it, instead of applying an upper bound with Device_max_PPM.

After the first successful UWB packet is received by vehicle, the vehicle should use this to resynchronize the UWB clocks between device and vehicle. Therefore, the vehicle should apply worst-case PPM based on the smallest duration between:

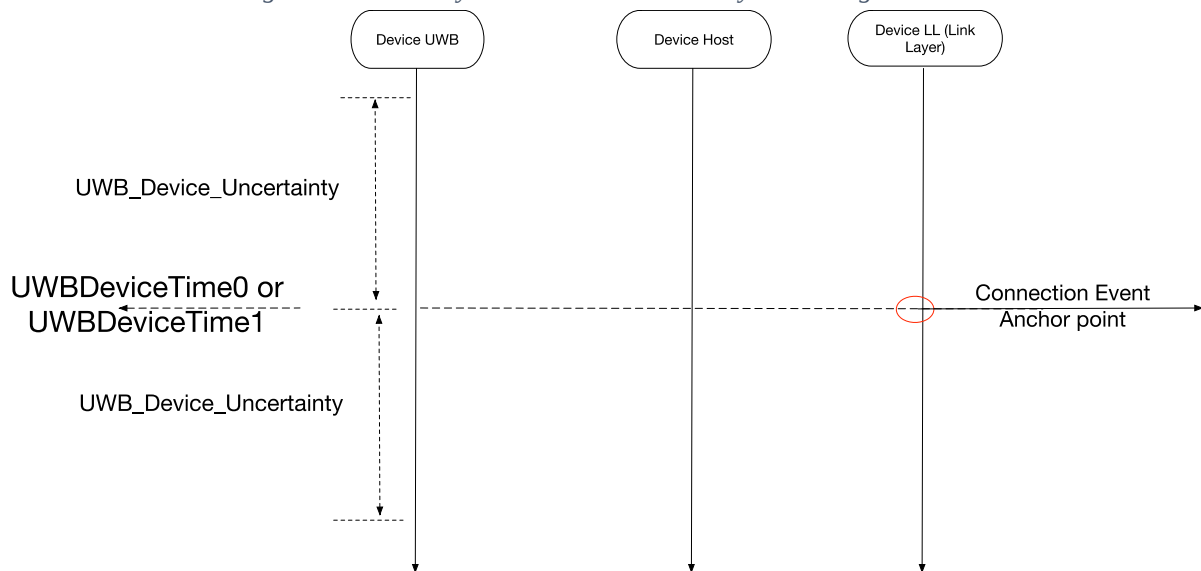
- Latest successful UWB packet exchange until now
- Latest time sync until now

19.4.3 Device Uncertainty

UWB_Device_Uncertainty is the time uncertainty between the connection event Anchor point and UWB Clock domain mapping (see Section 19.3.3 for definition)

Figure 19-8 shows the uncertainties on the device side.

Figure 19-8: Time Synchronization Uncertainty Flow Diagram.



19.4.4 “Bluetooth LE Timesync” Procedures.

In this subsection, the following times are defined:

- *UWBDeviceTime0*: UWB time clock domain captured at Procedure 0.
- *UWBDeviceTime1*: UWB time clock domain captured at Procedure 1.
- *DeviceEventCount0*: Event count for the connection interval after Bluetooth connection establishment. This is the first connection interval, and the value shall be 0. *UWBDeviceTime0* is taken at the anchor point of this connection interval.
- *DeviceEventCount1*: Event count for the connection interval containing the LL_PHY_UPDATE_IND message sent from device to the vehicle during the “Bluetooth LE TimeSync” procedure 1. *UWBDeviceTime1* is taken at the anchor point of this connection interval.

19.4.4.1 Procedure 0: Bluetooth LE Timesync at Bluetooth Connection

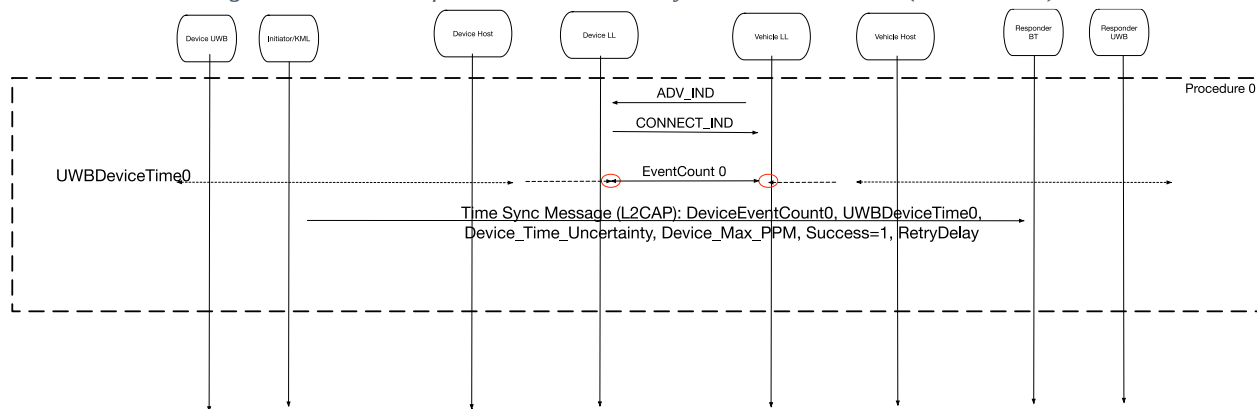
The host (device) shall capture the anchor point of the first available connection event after the Bluetooth connection establishment. The device shall map this point in time also into its UWB clock domain, (which corresponds to UWBDeviceTime0) and the device shall send a timesync message over Bluetooth LE. In case of retransmission for the connection event 0, the vehicle should consider only the latest successful LL message.

The timesync message is defined in Section 19.3.3.

The timesync message parameter DeviceEventCount should have a value of 0 or close to it.

A successful “procedure 0” shall return a timesync message parameter Success value of 1. The timesync message parameter RetryDelay has no meaning.

Figure 19-9: Successful Bluetooth LE timeSync at BT connection (Procedure 0).



19.4.4.2 Procedure 1: Bluetooth LE Timesync Triggered by Vehicle

When the vehicle determines a time synchronization is required, the host (vehicle) shall send an “LE Set PHY” command which triggers sending the “LL PHY REQ” to the device. The device should respond to the “LL PHY REQ” by sending “LL_PHY_UPDATE_IND”.

The vehicle may trigger the procedure 1 before or after the ranging session start. The vehicle shall trigger the procedure 1 only after Bluetooth connection is established and encryption is completed.

The time synchronization process on the device side is initiated once the device receives the message “LL_PHY_REQ” sent from the vehicle. The time synchronization shall happen at the anchor point of the connection event that contains the response message “LL_PHY_UPDATE_IND” of the device. This time as DeviceEventCount1.

The device shall map this point in time into its UWB clock domain (UWBDeviceTime1). The exact mechanism of mapping the timing reference is out of scope for this specification. In case of retransmission for “LL_PHY_UPDATE_IND”, the vehicle should consider only the latest successful “LL_PHY_UPDATE_IND” LL message.

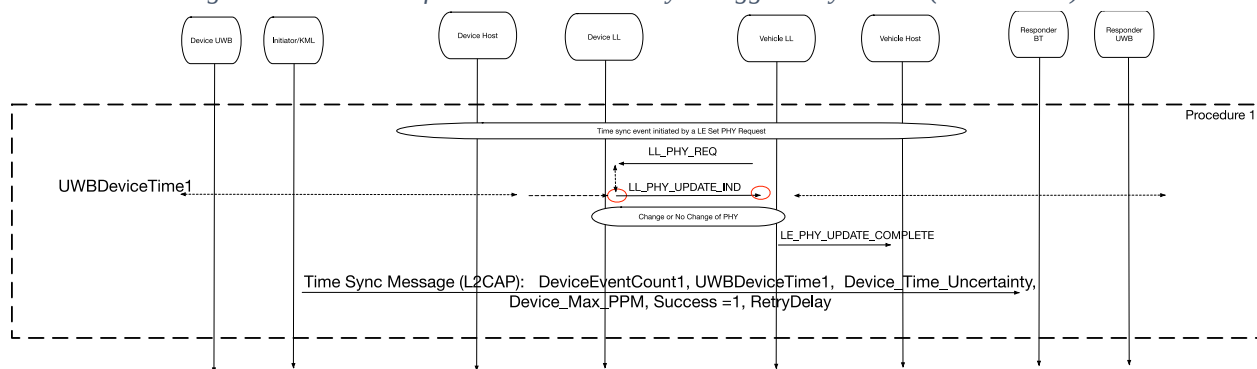
After mapping the timing reference into its UWB clock domain, the device shall send a timesync message over Bluetooth LE. The timesync message is defined in Section 19.3.3.

The vehicle should map the timing reference (anchor point) of a connection event into its UWB clock domain (UWBVehicleTime1).

The vehicle maps its UWB clock in two different ways:

1. **Same connection event as device:** Vehicle has mapped the anchor point of the same connection event as referenced in the timesync message.
 - a. In this case the vehicle may directly use UWB_Device_Time from the timesync message to determine the synchronization offset:
 - i. $\text{SyncOffset} = \text{UWBDeviceTime1} - \text{UWBVehicleTime1}$
 - b. The use of DeviceEventCount is not necessary.
2. **Different connection event as device:** Vehicle has mapped the anchor point of a different connection event VehicleEventCount1
 - a. In this case the vehicle shall use DeviceEventCount to correct for the different points in time and determine the synchronization offset:
 - i. $\text{SyncOffset} = \text{UWBDeviceTime1} + (\text{VehicleEventCount1} - \text{DeviceEventCount1}) \times \text{Connection Interval} - \text{UWBVehicleTime1}$
 - b. Clock skew between device UWB and BLE clock domain can be neglected, if the delta of the event counts for mapping is small.

Figure 19-10: Successful Bluetooth LE timeSync triggered by vehicle (Procedure 1).



19.4.4.3 Unsuccessful “Bluetooth LE Timesync”

An Unsuccessful “Bluetooth LE Timesync”, shown in Figure 19-11, is defined when the success field in the Time_Sync message (see Section 19.3.3.) is set to 0.

The device may return an unsuccessful “Bluetooth LE Timesync” for the following reasons:

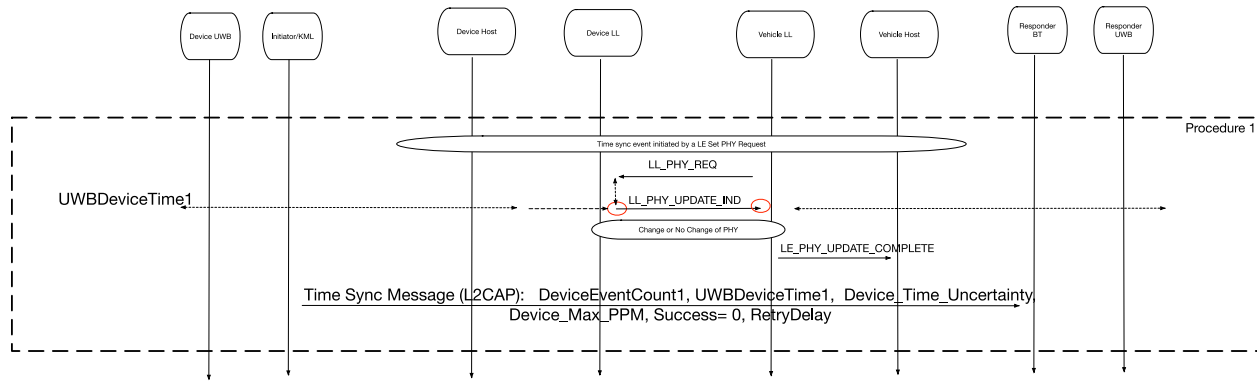
- Device UWB Clock is not available
- Device is unable to map the timing reference to its UWB clock domain.

If the success field is 0, the vehicle should initiate the “Bluetooth LE Timesync” procedure after the delay stated in the field “RetryDelay”.

In this case, the following parameters in the time Sync message are undefined or not available:

- UWB_Device_time
- DeviceEventCount1
- UWB_Device_time_uncertainty
- Device_max_PPM

Figure 19-11: Unsuccessful Timesync message.



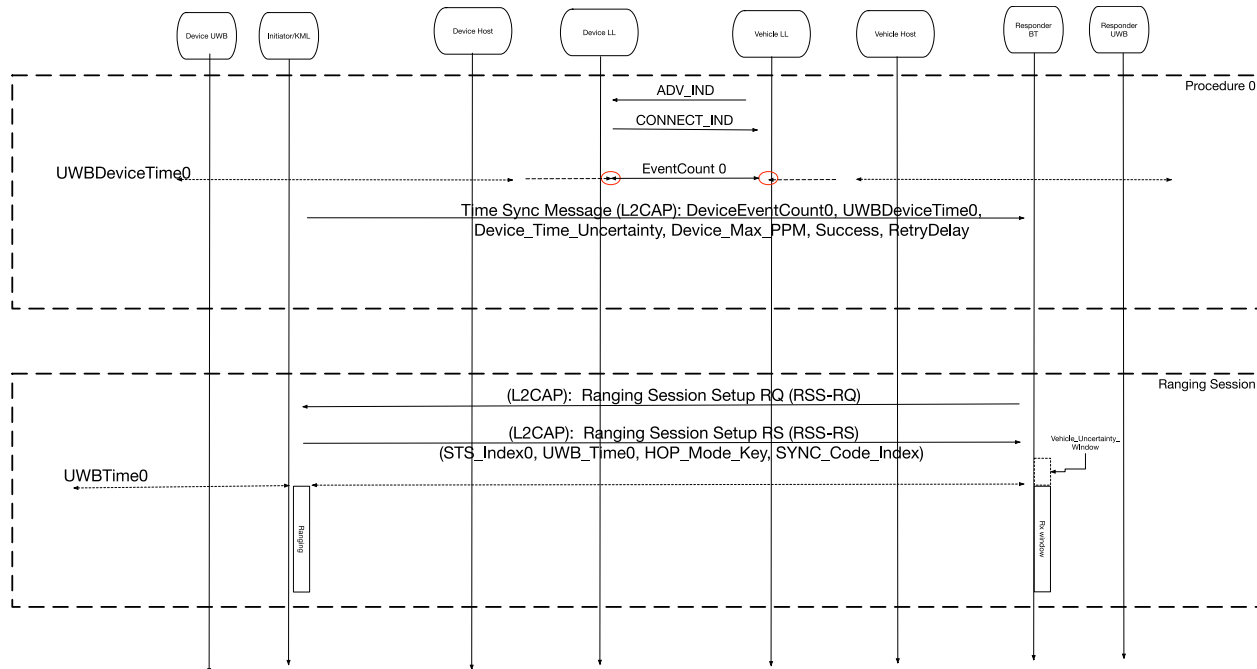
19.4.5 “Bluetooth LE Timesync” message flow examples

There are multiple combinations possible for Procedure 0 and Procedure 1 before a ranging session is setup.

19.4.5.1 Example 1: “Bluetooth LE Timesync” with Procedure 0 only

In Example 1, shown in Figure 19-12, the procedure 0 is successful and procedure 1 is omitted.

Figure 19-12: Bluetooth LE Timesync message flow example 1.



19.4.5.2 Example 2: “Bluetooth LE Timesync” with Procedure 0 and Procedure 1

In Example 2, both Procedure 0 and Procedure 1 are successful.

The internal operations of the device and vehicle for obtaining UWB timestamps according to the BLE timestamps are implementation specific.

Figure 19-13: Time Synchronization flow diagram example 2.

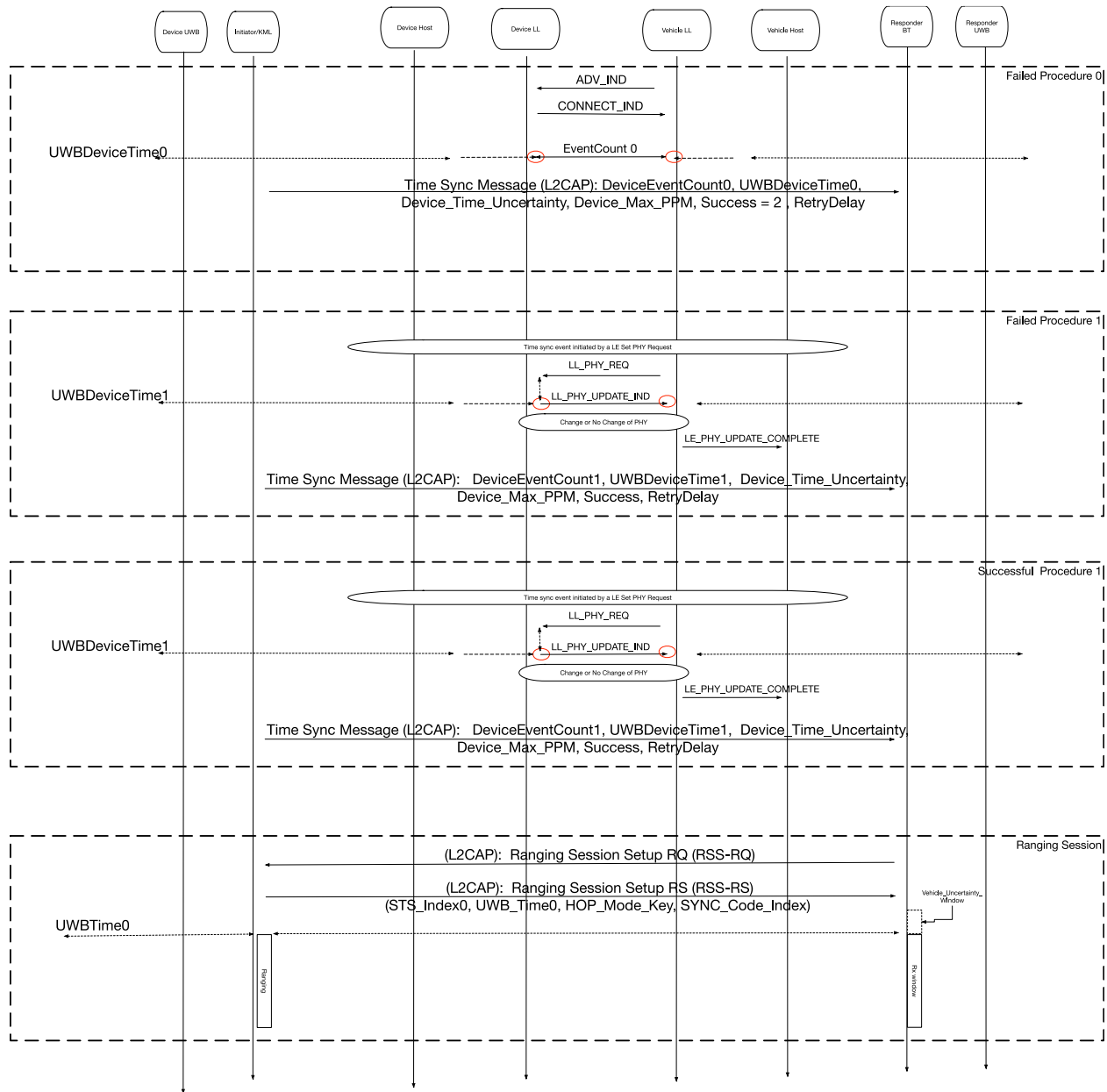


The second Procedure 1 shall be initiated after the retryDelay provided in the first Failed Procedure 1.

If the device supports Procedure 0, it should reply with Success = 2 (the “Bluetooth LE Timesync” procedure has failed and the Procedure 0 is not available) to avoid confusion for the vehicle.

1

Figure 19-14: Bluetooth LE Time Sync message flow example 3.



2

19.4.6 Conditions for a “Bluetooth LE Timesync” procedure

A successful “Procedure 1” is defined as the grouping of the following LL messages and DK messages:

- LL_PHY_REQ initiated by vehicle
- LL_PHY_UPDATE_IND sent by device
- Time_Sync message with success field = 1 (see Section 19.3.3).

Frequency of Procedure 1:

- If consecutive successful “Procedure 1” is equal or more than 60 seconds apart, the device shall support it.
- If consecutive successful “Procedure 1” is less than 60 seconds apart, device support is optional.

There should be at least one successful procedure before the Ranging Session Setup Request and Ranging Recovery Request messages.

UWB_Time0 (provided in the Ranging Session Setup Response or Ranging Recovery Response message) shall be consistent in time to the reference UWB_Device_Time. Since Device UWB clock enters "defined" mode after successful Procedure 1, any Bluetooth reconnection after the LL_PHY_UPDATE_IND and before the ranging shall not impact the UWB_Time0.

19.5 Digital Key – Flows

19.5.1 Owner Pairing Flow

Owner pairing can either be triggered automatically by the vehicle initiating Owner Pairing Connection Establishment as defined in Section 19.2.1 or manually by user going into the vehicle system menu and initiating owner pairing. In either case, before initiating owner pairing process, vehicle ensures that all of its policies have been met. Below are some of the examples for what these policies might include:

1. Physical key fob inside the vehicle
2. No owner device paired with vehicle
3. Digital Key feature has been enabled

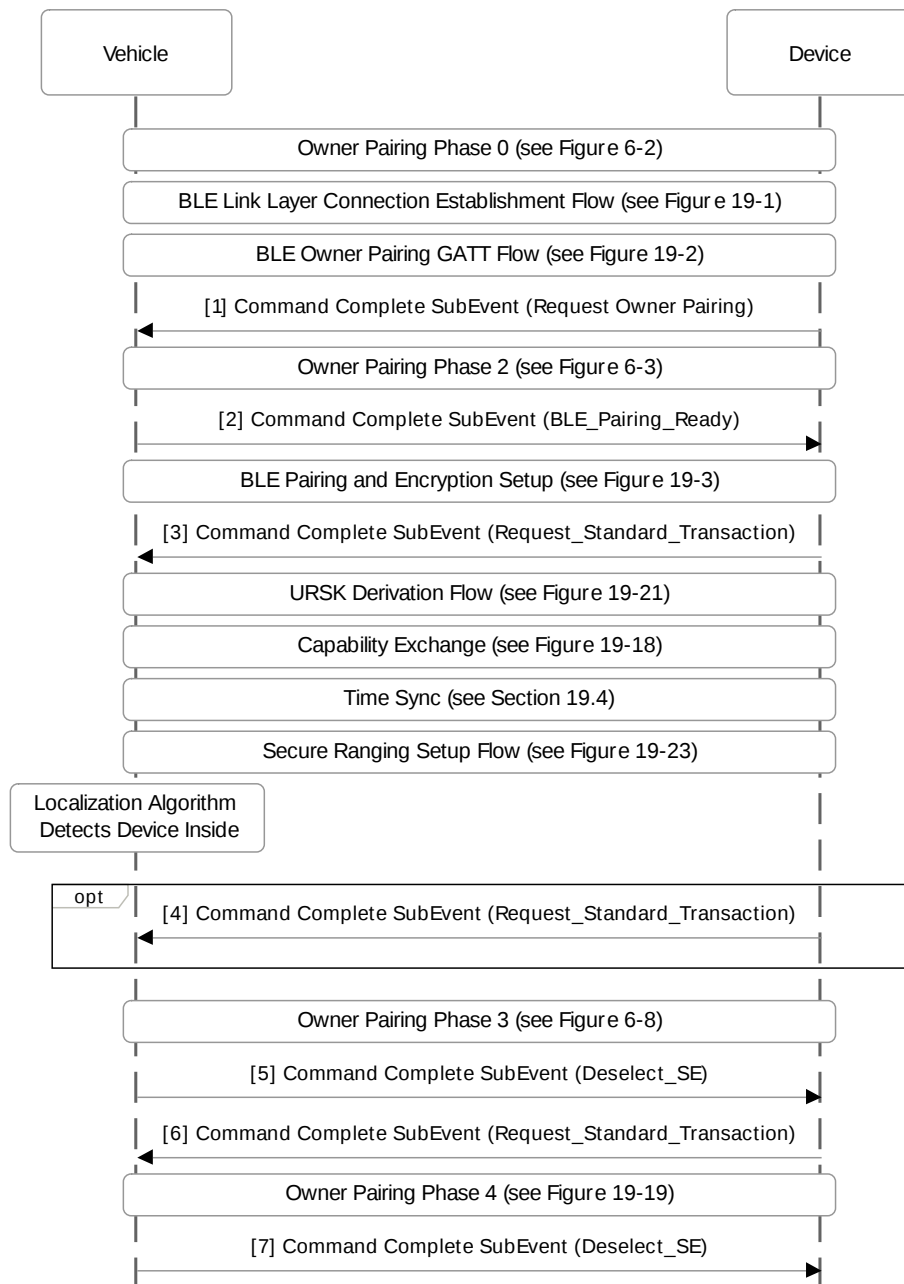
Owner pairing over Bluetooth LE leverages similar exchange that takes place over NFC with minor modifications and additional steps to the flow as described in this section and shown in Figure 19-15. NFC related procedures (NFC polling and link setup, NFC link teardown, NFC reset) in Owner Pairing Phase 2, Phase 3 and Phase 4 shall not apply to owner pairing over Bluetooth LE. For Bluetooth LE/NFC, the owner pairing may start over NFC as described in Section 6 and followed by Bluetooth LE Activation flow as described in Section 19.5.9.5. Owner pairing shall be performed over NFC if the vehicle or device does not support secure ranging.

When performing owner pairing over BLE only, the following rules apply:

1. On completion of BLE Pairing and encryption setup, (Figure 19-3) the device shall send a Request_Standard_Transaction subevent within $t_{O_{veh-5}}$.
2. If the vehicle does not receive a Request_Standard_Transaction within timeout $t_{O_{veh-5}}$, the vehicle shall initiate the Standard Transaction for URSK derivation.
3. After completion of Secure Ranging Setup Flow, the device may indicate readiness for Phase 3 by sending a Request_Standard_Transaction Subevent to the vehicle.
4. On completion on device localization, the vehicle shall initiate a Standard Transaction.
5. After Phase 3, on delivery of the KTS receipt from the Vehicle OEM server to the device, the device shall send a Request_Standard_Transaction Subevent to the vehicle.
6. If the vehicle does not receive a Request_Standard_Transaction subevent within $t_{O_{veh-5}}$, the vehicle shall initiate a Standard Transaction.

- 1 When performing Owner Pairing or Friend New Key Transaction (Section 19.5.8.2) over BLE (including
- 2 BLE only activation), if the vehicle initiates a Standard Transaction, and the device is not ready (e.g.,
- 3 KTS receipt not received), the device shall send a Device_busy subevent. (See Table 19-74)
- 4 If the vehicle receives a Device_busy subevent, to ensure that owner pairing or friend first approach over
- 5 BLE completes, the vehicle shall retry standard transaction initiation at regular intervals in the range of
- 6 $[T_{veh-loop}-0.5s, T_{veh-loop}+0.5s]$.

Figure 19-15: Owner Pairing Flow for Bluetooth LE/UWB.



Owner Pairing Phase 0: Preparation

This phase prepares device and vehicle for owner pairing as described in Section [6.3.1](#).

Bluetooth LE Link Layer Connection Establishment

In this setup process, the device goes through Bluetooth LE connection, discover DK services and characteristics and setup L2CAP channel as described above in Figure 19-1.

Bluetooth LE Owner Pairing GATT Flow

In this setup process, device goes through Bluetooth LE Owner Pairing GATT Flow as described in Figure 19-2.

Request Owner Pairing

After connection to the vehicles Bluetooth LE owner pairing advertisement, the device shall send the “Request_Owner_Pairing” SubEvent notification (see [Table 19-74](#)) to start owner pairing.

Owner Pairing Phase 2: SPAKE2+ Flow and Certificate Exchange

This phase enables SPAKE2+ based secure channel and certificate exchange between vehicle and device. For more details on Phase 2, refer to Section 6.3.3. To perform Phase 2 exchange over Bluetooth LE, each APDU command and response shall be encoded as below.

Message Type: Framework message

Message:

APDU Command: DK_APDU_RQ (see Section [19.3.2.1](#))

APDU Response: DK_APDU_RS (see Section [19.3.2.2](#))

In Owner Pairing Phase 2, both device and vehicle go through SPAKE2+ flow (see [Figure 6-3](#) and Figure 6-4) to derive system keys and optionally (if online BLE keys are not supported) OOB Bluetooth LE keys as defined below. In this version of Digital Key specification, the kble_intro and kble_oob_master don't need to be generated, instead the values are provided by the vehicle in tags D4_h and D5_h, respectively, in 7F49_h during owner pairing. More details on SPAKE2+ and system keys derivation (HKDF functions) see Section [18](#).

Below is the set of derived system keys.

```
Keys = HKDF-SHA256(96, K, "SystemKeys")           // (output_len,  
                                                    // input_keying_material, info)  
Kenc = Keys[0:15]                                  // first 16 bytes  
Kmac = Keys[16:31]                                 // second 16 bytes  
Krmac = Keys[32:47]                                // third 16 bytes  
LONG_TERM_SHARED_SECRET = Keys[48:63]             // fourth 16 bytes  
Kble_intro = Keys[64:79]                           // fifth 16 bytes  
Kble_oob_master = Keys[80:95]                      // sixth 16 bytes
```

Out of these keys, Kble_oob_master and Kble_intro are UWB solution specific keys:

1. Kble_oob_master is used as shared secret between device and vehicle which is used to derive Kble_oob to encrypt OOB pairing data during First Approach
2. Kble_intro is used to encrypt DK_Identifier during First Approach

In addition, the vehicle shall provide its “Wireless Capabilities” as part of 7F49 template defined in Table 19-90. The following wireless capability combinations (WCC) as utilized for DK Vehicle and Device operation are defined:

- WCC1: NFC only
- WCC2: NFC and Bluetooth LE (NFC + RKE Functions)
- WCC3: NFC, Bluetooth LE, and UWB (NFC + RKE Functions + Passive/Location-based Functions)

Devices and Vehicles may contain any combination of radios not used for DK operations.

Device shall store Kble_oob_master, Bluetooth Random Static Address of the vehicle and IRK of vehicle in its database. This information shall be used during owner pairing, NFC to UWB migration of owner device and key sharing.

If the vehicle supports online retrieval of BLE key material, kble_intro and kble_oob_master shall be synchronized between the vehicle and vehicle backend prior to the owner pairing. The method to achieve is out of scope of the specification.

If a shared key receives BLE keys in Key Tracking Response then kble_oob and kble_intro received from the sender device shall be overwritten with the values in Key Tracking Response.

Table 19-90: 7F49 Template.

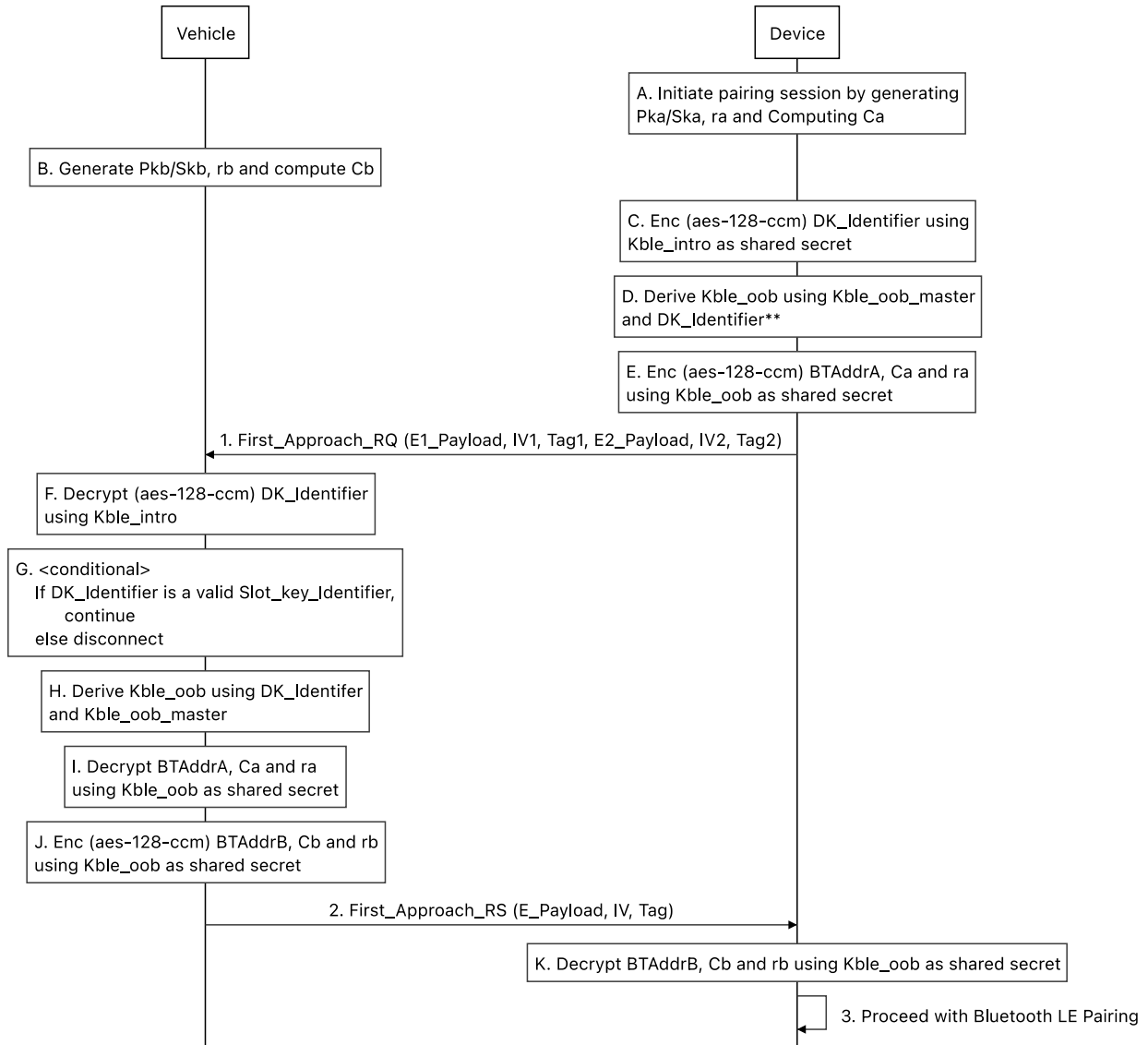
Template			Length (bytes)	Description	Field is												
7F49 _h			variable	7F49 template is used to provide wireless capability (NFC/UWB) and related data; this template shall be included by vehicle.	M												
	5F50 _h		0	This tag shall be included if and only if the vehicle supports NFC. Vehicle and Device shall always support NFC capability. If this tag is present, tags 5F51 and/or 7F52 may be present.	M												
	5F51 _h		0	This tag shall be included if and only if the vehicle supports UWB. If this tag is present, tags 5F50 and 7F52 shall be present.	O												
	7F52 _h		variable	This tag (Bluetooth LE) shall be included if and only if either of the WCC shown below are supported by the vehicle:<table><tr><th>WCC</th><th>NFC</th><th>UWB</th><th>BLE</th></tr><tr><td>WCC2</td><td>Supported</td><td>Not Supported</td><td>Supported</td></tr><tr><td>WCC3</td><td>Supported</td><td>Supported</td><td>Supported</td></tr></table> Length for usage during Owner pairing: 28 bytes Length for usage during Key sharing: 64 bytes	WCC	NFC	UWB	BLE	WCC2	Supported	Not Supported	Supported	WCC3	Supported	Supported	Supported	C
WCC	NFC	UWB	BLE														
WCC2	Supported	Not Supported	Supported														
WCC3	Supported	Supported	Supported														
		D0 _h	16	IRK of Bluetooth LE of the vehicle. If Tag 7F52 _h is present, this tag shall be included during Owner Pairing (See Table 5-14) and in an Import Request (See Table 11-11) sent during key sharing. This tag shall not be included in key creation request (see Table 11-5).	C												
		D1 _h	6	Bluetooth Random Static Address of the vehicle. If Tag 7F52 _h is present, this tag shall be included during Owner Pairing (See Table 5-14) and in an Import Request (See Table	C												

Template			Length (bytes)	Description	Field is
				11-11) sent during key sharing. This tag shall not be included in key creation request (see Table 11-5).	
		D2 _h	0	This Tag shall be included if vehicle supports LELR.	O
		D3 _h	16	Kble_oob is a 16-byte key used to encrypt/decrypt (AES-128-CCM) Bluetooth LE Secure OOB Pairing data which is exchanged as part of First_Approach_RQ/RS. This key is derived by owner device and vehicle and unique per Digital Key. This tag shall not be included during owner pairing (See Table 5-14), when transferring the “Wireless capabilities” from vehicle to device or with Key Creation Request (see Table 11-5). This tag shall only be included in an Import Request (See Table 11-11) sent during key sharing. If the vehicle, owner device, and receiver device support the online retrieval of BLE key material (agreed OD-FD-KS equal to 0300_h or higher), during key sharing, then tag D3_h may not be present during sharing.	C
		D4 _h	16	Kble_intro is a 16-byte key used to encrypt/decrypt (AES-128-CCM) DK_Identifier exchanged as part of First_Approach_RQ. This key is derived by both device and vehicle as part of owner pairing flow (see Section 19.5.1) and is same across all receivers. This tag shall not be included during owner pairing (See Table 5-14), when transferring the “Wireless capabilities” from vehicle to device or with Key Creation Request (see Table 11-5). This tag shall only be included in an Import Request (See Table 11-11) sent during key sharing. If the vehicle and owner device agreed on the online retrieval of BLE-Material, during owner pairing tag D4_h shall be present and contain Kble_intro as provisioned to vehicle by vehicle OEM backend (If the device has already derived Kble_intro from PAKE, it shall be overwritten with the value provided in tag D4_h). If during owner pairing tag 0xD4 is not present, device shall use Kble_intro derived from PAKE. If the vehicle, owner device, and receiver device support the online retrieval of BLE key material (agreed OD-FD-KS equal to 0300_h or higher), during key sharing, then tag D4_h may not be present during sharing.	C
		D5 _h	16	If the vehicle and the owner device agreed on the online retrieval of BLE key material (5B_h equal to 0300_h or higher), during owner pairing, then D5_h shall be present and contain kble_oob_master as provisioned by vehicle. This tag shall only be used for owner pairing. If the device has already derived kble_oob_master from SPAKE2+, it shall be overwritten with the value provided in D5_h. If during owner pairing D5_h is not present, device shall use kble_oob_master derived from SPAKE2+.	C

- 1 Additional tags sent by the vehicle in 7F49 template shall be ignored by the device.
- 2
- 3 The vehicle that supports Bluetooth LE/UWB shall also set the corresponding bits in Tag 46_h and
- 4 Tag 47_h as defined in Table 15-14.
- 5 **Bluetooth LE Pairing & Encryption Setup Procedure**
- 6 The Bluetooth LE Secure OOB Pairing Prep flow, shown in Figure 19-16, enables secure sharing
- 7 of OOB data between device and vehicle to enable Bluetooth LE Secure OOB pairing.
- 8 **Message Type:** Supplementary Service message
- 9 **Message:**

First_Approach_RQ (see Section 19.3.4.1)
First_Approach_RS (see Section 19.3.4.2)

Figure 19-16: Bluetooth LE Secure OOB Pairing Prep.



** A receiver device does not derive Kble_oob but receives it from the Owner device instead.

Elements of the Bluetooth LE Secure OOB Pairing Prep flow:

- The BTAAddrA and BTAAddrB are the Device Bluetooth Address and Vehicle Bluetooth Address respectively, used during pairing (See Vol 3, Part H, Section 2.2.7 of [30]).
- For box F, vehicle shall use AES-128-CCM with Kble_intro as shared secret and 8-byte IV provided by device to decrypt DK_Identifier
- For box G, vehicle shall react depending on Tag DAh configuration in Table 5-14.

If Tag DA_h in Table 5-14 is not present, or set to values 00_h or 02_h, vehicle shall validate that decrypted DK_Identifier value belongs to a valid slot.

If Tag DA_h in Table 5-14 is present and set to values 01_h, 03_h, 04_h, 05_h, vehicle shall not validate that decrypted DK_Identifier value belongs to a valid slot in this step. The vehicle shall validate the slot identifier during the first standard transaction after Bluetooth LE pairing instead.

- For box H, vehicle shall use HKDF function with decrypted DK_Identifier and Kble_oob_master to derive Kble_oob.
 - Kble_oob = HKDF-SHA256(16, Kble_oob_master, DK_Identifier) // (output_len, input_keying_material, info)
 - Where DK_Identifier is an 8-byte padded slot_identifier. For example, if the slot_identifier is 6-byte, it will have 0x0000 added as prefix to make it 8-byte.
- For box I, vehicle shall use AES-128-CCM with Kble_oob as shared secret and 8byte IV to decrypt device OOB data required for Bluetooth LE Secure OOB pairing (BTAddrA || Ca || ra)
- For box J, vehicle shall use AES-128-CCM with Kble_oob as shared secret and 8-byte IV to encrypt vehicle OOB data required for Bluetooth LE Secure OOB pairing (BTAddrB || Cb || rb)
- For box K, device shall use AES-128-CCM with Kble_intro as shared secret and 8-byte IV provided by device to decrypt DK_Identifier
- Once device and vehicle have successfully performed Bluetooth LE Secure OOB Pairing Prep, device shall initiate Bluetooth LE Pairing procedure by sending pairing request as described in Section [19.2.1.4](#). Post successful pairing, device shall setup Bluetooth LE encryption before moving to next step.
- The Bluetooth LE Pairing data shall be persisted for device and vehicle at the same point in time the endpoint is persisted on either side. Any failure before that point shall lead to rolling back the Bluetooth LE pairing data to be able to perform a retry.
- Any Bluetooth LE messages being exchanged over-the-air as part of this flow shall not make use of DK message defined in Section [19.3](#).

URSK Derivation Flow

Once the Bluetooth LE encryption has been setup, if the vehicle supports UWB, the vehicle shall execute the standard transaction flow for deriving URSK. (The URSK Derivation flow is shown in Figure 19-20)

To derive URSK over Bluetooth LE, each APDU command and response shall be encoded as shown below.

Message Type: SE message

Message:

APDU Command Encapsulation: DK_APDU_RQ (see Section [19.3.2.1](#))

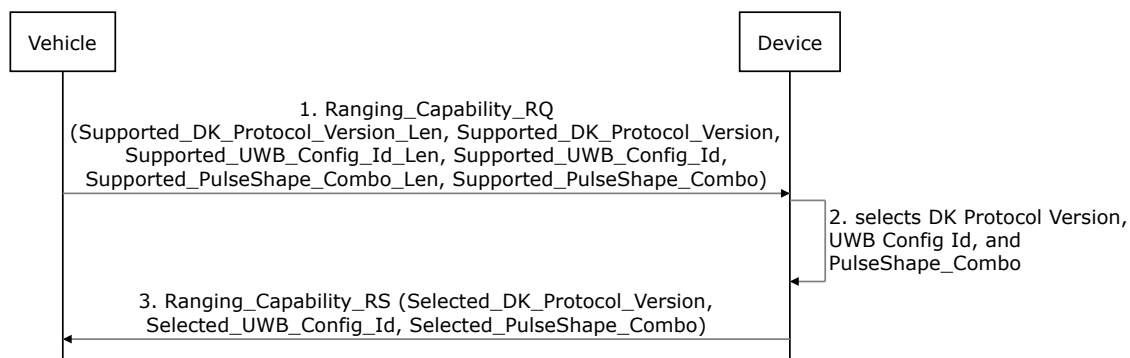
APDU Response Encapsulation: DK_APDU_RS (see Section [19.3.2.2](#))

Capability Exchange

Capability Exchange for owner pairing shall be performed if either the vehicle or device has an updated DK Protocol Version, UWB Configuration Identifier, PulseShape Combinations, or a combination of these parameters, which is different than the one the vehicle and device supported during the last Capability Exchange. The Capability Exchange is shown in Figure 19-17.

The vehicle initiates the Capability Exchange by sending Ranging_Capability_RQ (see Section 19.3.1.1) to the device. Upon receiving the Ranging_Capability_RQ, the device shall respond by sending Ranging_Capability_RS (see Section 19.3.1.2).

Figure 19-17: Capability Exchange.



The device triggers the Capability Exchange by sending DK Event Notification with Subevent_Category set to 0x01_h and the corresponding command status code set to 0x02_h. Upon receiving the DK Event Notification, the vehicle shall initiate the Capability Exchange as shown in Figure 19-17.

Time Sync

If the Device UWB Clock is “Not in sync” while Bluetooth LE connected with the device, the vehicle shall trigger the Bluetooth LE Timesync procedure 1 if all conditions are met (see Section 19.4.2).

Secure Ranging Setup Flow

Once the Capability Exchange is done, vehicle shall initiate secure ranging setup flow. In this flow, the vehicle and device go through a few exchanges to finalize the ranging parameter values required to set up the secure ranging. As part of this flow, the vehicle shall notify the device on which URSK to use by sending Ranging_Session_RQ with its UWB_Session_Id.

To set up secure ranging over Bluetooth LE, each message shall be encoded as below:

Message Type: UWB Ranging Service message

Message:

Ranging Service Message: RS_RQ (see Section 19.3.1.3)

Ranging Service Message: RS_RS (see Section 19.3.1.4)

Ranging Service Message: RSS_RQ (see Section 19.3.1.5)

Ranging Service Message: RSS_RS (see Section [19.3.1.6](#))

DK Event Notification (see Section [19.3.9](#))

Figure 19-23 illustrates the Secure Ranging Setup flow.

The vehicle shall perform secure ranging and detect that the device is inside the vehicle before proceeding to Owner Pairing Phase 3. If the device is not detected inside of the vehicle or other ranging failure occurred, the vehicle shall send an appropriate SubEvent code defined in Table 19-73 or Table 19-75 and abort the owner pairing attempt.

Owner Pairing Phase 3

To perform Owner Pairing Phase 3 exchange (see Section [6.3.4](#)) over Bluetooth LE, each APDU command and response shall be encoded as below.

Message Type: SE message

Message:

APDU Command Encapsulation: DK_APDU_RQ (see Section [19.3.2.1](#))

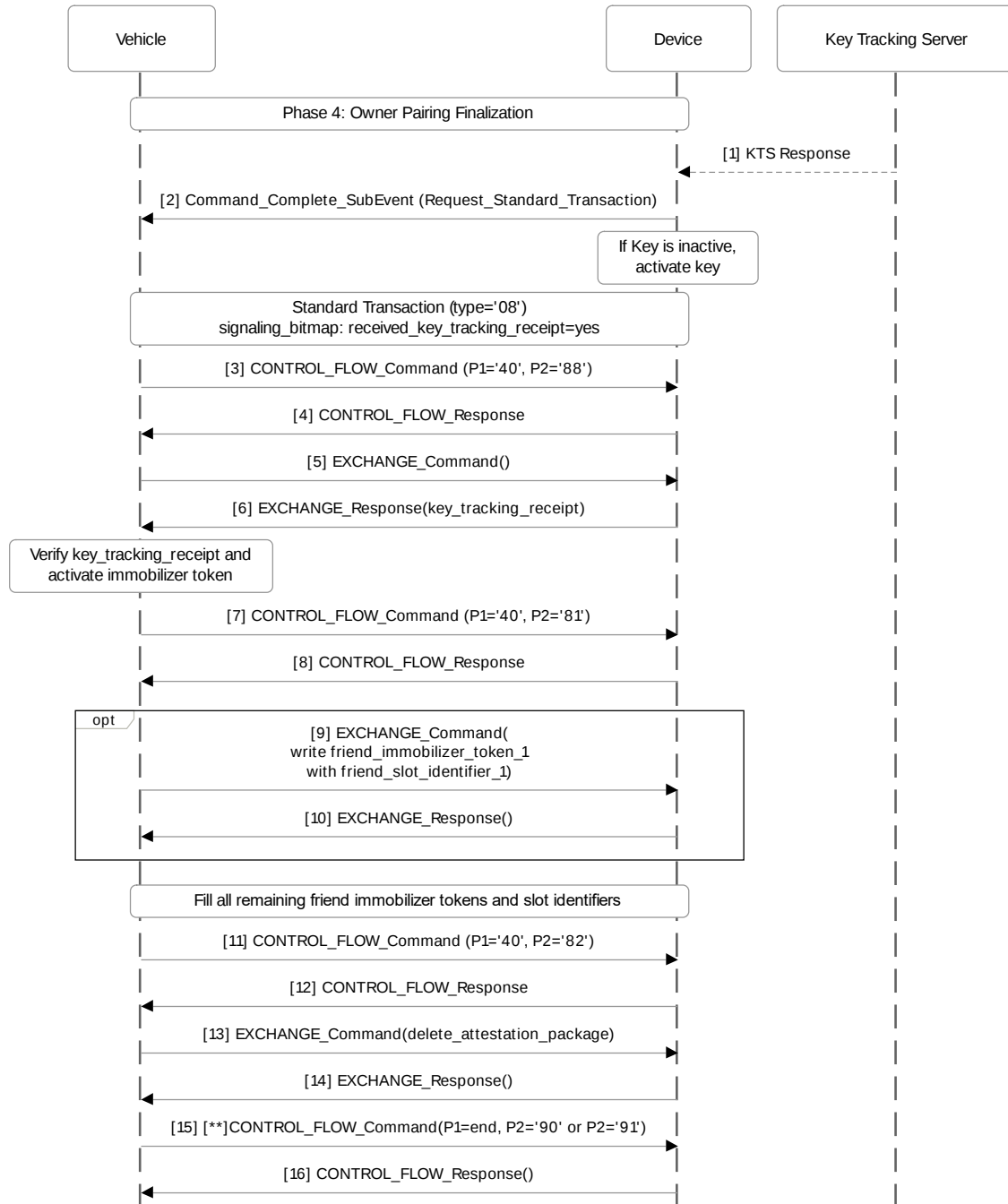
APDU Response Encapsulation: DK_APDU_RS (see Section [19.3.2.2](#))

Owner Pairing Phase 4

Owner Pairing Phase 4 exchange (see Section [6.3.5](#)) over Bluetooth LE shall be modified as shown in Figure 19-18. If KTS is supported, the device shall wait for KTS the response before sending the Request_standard_transaction subevent to the vehicle to trigger standard transaction.

1

Figure 19-18: Owner Pairing Phase 4 over Bluetooth LE



2

3 Each APDU command and response shall be encoded as below

4 **Message Type:** SE message

5 **Message:**

6 APDU Command Encapsulation: DK_APDU_RQ (see Section [19.3.2.1](#))

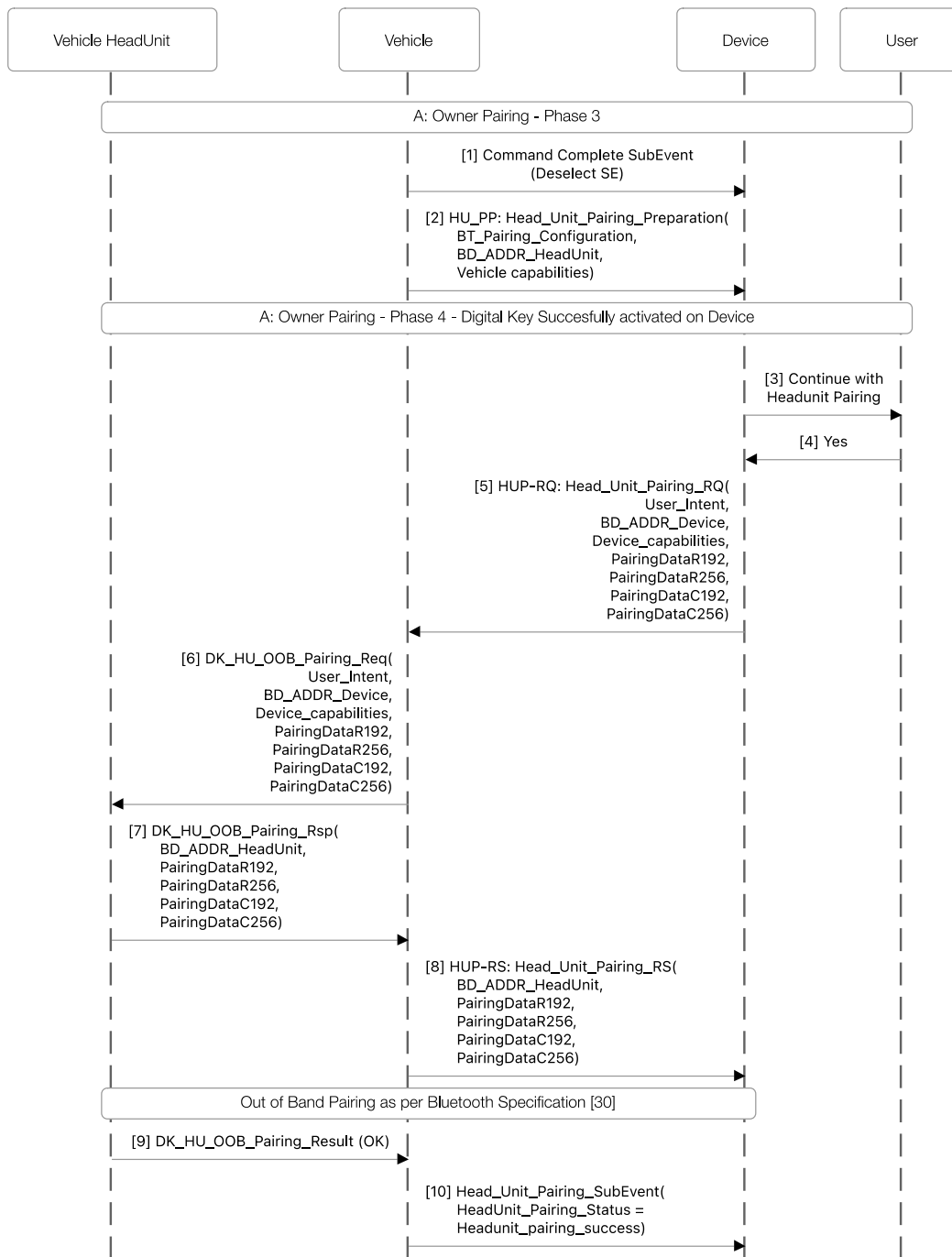
7 APDU Response Encapsulation: DK_APDU_RS (see Section [19.3.2.2](#))

1 19.5.2 Head Unit Pairing Flow

2 This section describes the Head Unit Pairing flow once the DK pairing is completed. This
3 feature, referred as “Head Unit Pairing”, is optional for both the device and vehicle (See [Table](#)
4 [19-15](#)). This feature only applies if the device’s Bluetooth supports BR/EDR (classic), and the
5 vehicle Bluetooth module for DK is different from that of the head unit (i.e. in a different
6 subsystem or a physical implementation). The Head Unit Pairing flow is shown in [Figure 19-19](#).
7 The Head Unit Pairing Preparation (HU-PP) message is used for head unit capability discovery
8 and initiating the Head Unit Pairing.
9 The vehicle may send HU-PP either before or after owner pairing is completed. Sending the HU-
10 PP prior to owner pairing completion allows the device to display a single message at the
11 Subevent Notification about DK pairing complete and head unit capability. If the Head Unit
12 Pairing is supported by the vehicle, the vehicle shall send the HU-PP within 2 seconds from the
13 Command Complete Subevent (Deselect_SE) at the end of owner pairing Phase 4 i.e., step 5 in
14 [Figure 19-15](#). If the Head Unit Pairing is not supported or for all subsequent head unit (re)pairing
15 the device and vehicle shall follow the Bluetooth pairing and encryption setup as defined in [\[30\]](#).

1

Figure 19-19: Combined Head Unit Pairing Flow



2

- 3 Note that the order of HU-PP message and owner pairing subsection (refer to Box A in Figure
4 19-19) is an example and can be interchanged.
5 If a device supports Head Unit Pairing, the device shall send HUP-RQ in step 5 (see Section
6 [19.3.8.2](#)). The vehicle shall respond with HUP-RS in step 8 (see Section [19.3.8.3](#)) to complete
7 the Head Unit Pairing. If the device does not support Head Unit Pairing, the flow in [Figure 19-19](#)
8 completes after step 2 and subsequent steps are not required.

If Headunit pairing (step 10) is not finished within 60 seconds from the moment the message HUP-REQ has been sent, the device may retry the head unit pairing from step 3.

Note that DK_HU_OOB_Pairing_Req and DK_HU_OOB_Pairing_Result messages in step 6 and 7 are implementation-specific by the vehicle OEM and are only provided as an example. For First New Key Transaction (Section [19.5.8.2](#)), the message HU-PP shall be sent after the “secure ranging setup flow” is completed.

The flow diagram in [Figure 19-19](#) shows the case where OOB communication is possible from both directions. The vehicle shall configure the parameter BT_Pairing_Configuration in message 1 to indicate to device that the vehicle head unit supports bidirectional OOB communication. Unidirectional OOB communication is not permitted.

19.5.2.1 Head Unit Pairing Flow (Owner Pairing over BLE)

This section describes the Head Unit Pairing flow in relation to owner pairing flow over BLE. If HUP is supported by the vehicle, the process shall be started by the vehicle sending HU_PP immediately after a successful phase 3, more precisely after the Command Complete Subevent (Deselect_SE).

The vehicle should only send HU_PP when the device is detected inside the vehicle and the HU infrastructure including BT is available and ready for HU pairing.

The vehicle may limit the time in which a HUP_RQ of a specific device is accepted to a configurable, suitable value after the HU_PP command was sent (e.g., 30 seconds). If no HUP_RQ is received within this time after a HU_PP command, the Head Unit (HU) may attempt another pairing in another drive cycle when the preconditions are met. After sending the HUP_RQ command, vehicle and device shall monitor the duration of the HU pairing process and abort the process after a configurable timeout (e.g., 15 seconds). During the HU pairing process further HUP_RQ commands might be ignored by the vehicle to avoid other devices or timeout discrepancies to interfere with the pairing process. During this time HU_PP messages to other devices might be delayed until the running process with the current device is finished.

If the pairing process is aborted after reaching the time limit, the device may instantly retry head unit pairing, by resending HUP_RQ again using the stored user consent.

If HU_PP data (Tag 9F20_h) is included in device configuration (Tag 7F4E_h), device can initiate the user consent flow at end of Phase 4. Therefore, vehicle and device may complete Head unit pairing even after NFC owner pairing if the device and vehicle support BLE for digital key.

In all cases the user consent for HU pairing might be requested at any point in time during the owner pairing UI flow after the HU_PP message has been received by the device.

Note: If the Head Unit Pairing as described in this specification is not supported the vehicle and device may follow the Bluetooth pairing and encryption setup as defined in [\[30\]](#).

For all subsequent head unit (re)pairing the device and vehicle shall follow the Bluetooth pairing and encryption setup as defined in [\[30\]](#).

If a device supports Head Unit Pairing, the device shall send HUP_RQ message in step 5 (see Section [19.3.8.2](#)). The vehicle shall respond with HUP_RS in step 8 (see Section [19.3.8.3](#)) to complete the Head Unit Pairing. If the device does not support Head Unit Pairing, the flow in [Figure 19-19](#) completes after step 2 and subsequent steps are not required.

If phase 3 or phase 4 cannot be completed during BLE owner pairing (e.g., device not connected to internet during owner pairing), then the vehicle executes the FNKT flow for the owner device on next approach as per Section 19.5.8.2. The message HU_PP shall be sent after the “secure ranging setup flow” is completed, the device is located inside the vehicle and the HU is ready for the pairing process.

Note that DK_HU_OOB_Pairing_Req and DK_HU_OOB_Pairing_Result messages in step 6 and 7 are implementation-specific by the vehicle OEM and are only provided as an example.

19.5.2.2 Head Unit Pairing Flow (Owner Pairing over NFC)

If HU_PP data (Tag 9F20_h) is included in device configuration (Tag 7F4E_h), device can initiate the user consent flow at end of Phase 4. Therefore, vehicle and device may complete Head unit pairing even after NFC owner pairing if the device and vehicle support BLE for digital key.

19.5.2.3 Head Unit Pairing Flow (Shared Key)

In order for the friend device to know whether the vehicle supports HUP, the content of the HU_PP message shall be provided in the key creation request (See Table 11-5). In this way, steps 3 and 4 (Figure 19-19) might be done when the sharing invitation is accepted on the friend device.

For First New Key Transaction (Section 19.5.8.2), the message HU-PP shall be sent after the “secure ranging setup flow” and if the device is correctly located inside the vehicle, the HU is ready for the pairing process and if the Device has supports HU-Pairing according to Bit[3] in Table 19-15.

NOTE: The HU-Pairing can be initiated by the car for example at a following engine start.

The vehicle should send the HU_PP message only to devices not completed Head Unit Pairing. As a fallback, if a device receives a HU_PP message even if the HU is already paired, the device shall ignore this message and should send the HUP_RQ message with a cancelation HU pairing status as follows:

User Intent = “no intent from device”, bit0 = 1.

The flow diagram in Figure 19-19 shows the case where OOB communication is possible from both directions. The vehicle shall configure the parameter BT_Pairing_Configuration in message 1 to indicate to device if the vehicle head unit supports bidirectional OOB communication.

If OOB communication is possible only in the direction from vehicle head unit to device, the parameters PairingDataR192, PairingDataR256, PairingDataC192, and PairingDataC256 in step 6 should be ignored by the vehicle head unit (see volume 2 Part H Section 7.2.2 of [30]).

19.5.3 Passive Entry

After owner pairing has been successfully performed, upon each approach, at minimum secure ranging flow shall be exercised to establish secure ranging before enabling any of the passive entry features/actions such as ‘Welcome Lights’, ‘Unlock’, ‘Lock’, etc. Once the secure ranging has been established and device has been localized, vehicle may decide to initiate any of the above actions, as per its policy or requirements.

For passive entry, one of the following conditions shall trigger the vehicle (except if the vehicle is in low power mode, or vehicle is not ready, for example, due to internal conflict) to establish secure ranging with the device:

1. Device sends Device Ranging Intent SubEvent with at least low_approach_confidence.
 2. The vehicle determines that there is user intent to use vehicle features that require secure ranging with the device, e.g., user touches door handle
- The URSK is needed before secure ranging can be established. The vehicle may have a pre-derived URSK or derive a new URSK as needed. The URSK confidentiality and integrity shall be preserved during the whole lifetime of the URSK.

19.5.4 URSK Management

Vehicle may derive and store a new URSK by:

- Executing a dedicated transaction flow (i.e. with a AUTH0 command indicating a transaction_code value 10_h) followed by command "CREATE RANGING KEY" as shown in [Figure 19-20](#).
OR
- By adding a CREATE RANGING KEY command to a Standard Transaction flow intended for another purpose like start engine (i.e., with a AUTH0 command indicating a transaction_code value not equal to 10_h) as shown in [Figure 19-30](#); see section [19.5.6.1](#). Each time this flow is executed successfully, a unique URSK is derived and stored indexed using a UWB_Session_Id. This UWB_Session_Id is the least significant 4 bytes of the transaction_identifier, a 16 bytes random number generated by the vehicle for each AUTH0 and sent to the device as part of AUTH0 command along with other parameters. For more details, see Section 15.3.2.9.

To derive URSK over Bluetooth LE, each APDU command and response shall be encoded as below.

Message Type: SE message

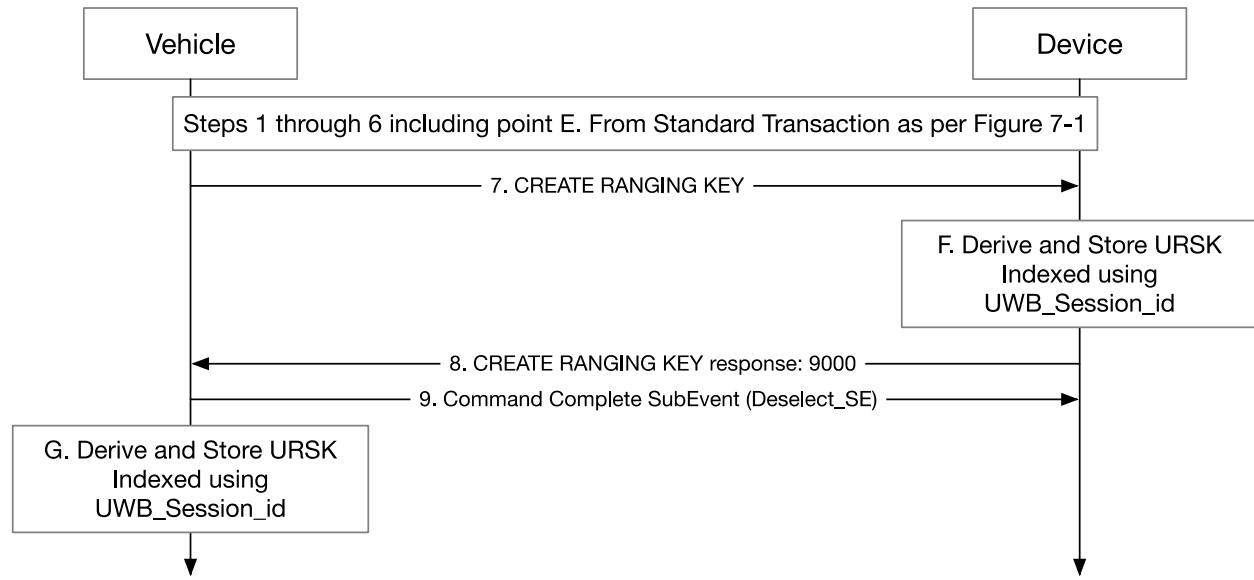
Message:

APDU Command Encapsulation: DK_APDU_RQ (see Section [19.3.2.1](#))

APDU Response Encapsulation: DK_APDU_RS (see Section [19.3.2.2](#))

1

Figure 19-20: URSK Derivation Flow in a dedicated Standard Transaction

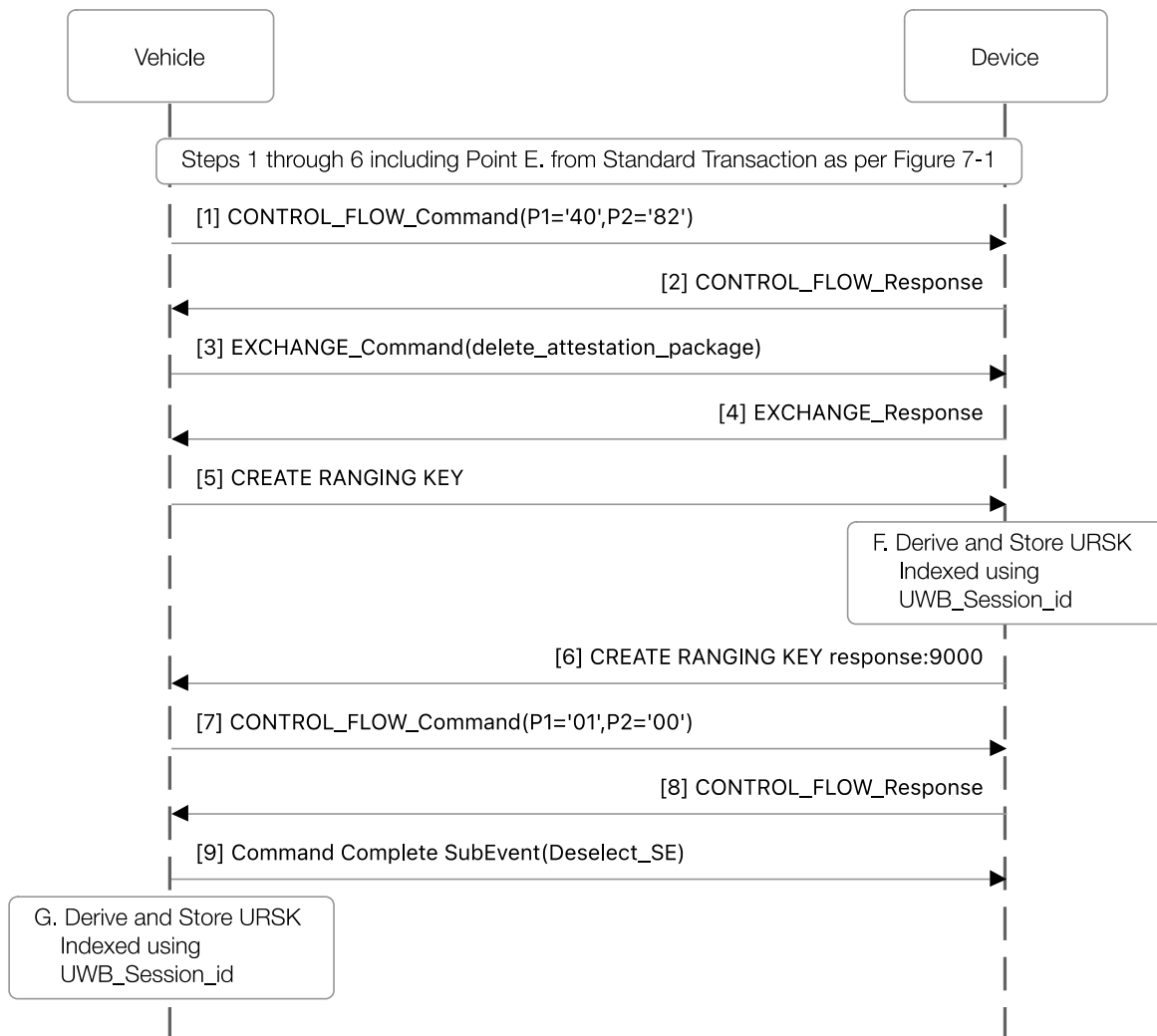


2

3

1

Figure 19-21: Create Ranging Key Transaction used to clear Private Mailbox



3

4 To find details on vehicle's processing for box A, E and H, see Section 7 and Section 15.3.2.26.

5 For box G and H, once the URSK has been successfully derived, it shall be stored in secure
6 storage indexed using UWB_Session_Id.

7 Vehicles capable of secure ranging shall support the derivation and storage of at least one pre-
8 derived URSK per Digital Key. Devices capable of secure ranging shall support the derivation
9 and storage of two pre-derived URSKs per Digital Key. Each of these pre-derived URSKs is
10 indexed using its associated UWB_Session_Id.

11 The vehicle shall use the Secure Ranging Setup Flow (see Figure 19-23) to activate a pre-derived
12 URSK. There shall be at most one active URSK per Digital Key, and this active URSK is
13 discarded anytime another one is activated.

14 After a pre-derived URSK is activated, the vehicle shall request derivation and storage of another
15 pre-derived URSK. The vehicle shall control when this additional URSK derivation occurs to
16 minimize user impact (for example, after passive entry completed) but is recommended to occur
17 shortly after URSK activation. Table 19-91 shows the URSK storage requirements.

1 *Table 19-91: URSK storage requirements per Digital Key endpoint.*

URSK Types	Definition	Vehicle URSK Storage	Device URSK Storage
Pre-derived	URSK that has not been activated using the Secure Ranging Setup Flow and has only been kept in secure storage. TTL is not defined until key is activated.	At least 1	2
Active	URSK activated using the Secure Ranging Setup Flow and has neither exhausted its STS_Index max incrementation nor has an expired TTL.	1	1

2 The URSK shall be discarded if one of the following conditions is met:

- 3 1. The STS_Index reaches its maximum value of $2^{31}-1$.
- 4 2. The STS_Index was lost, and it cannot be ensured that a previously used STS_Index
- 5 won't be used again
- 6 3. The URSK TTL (Time-To-Live) has expired. The vehicle shall enforce an URSK TTL
- 7 lower or equal to 12 hours (vehicle OEM specific). The URSK TTL starts when the first
- 8 dURSK is derived. The first dURSK shall be derived immediately before sending or after
- 9 receiving Ranging Session Setup Response for the device or vehicle, respectively.
- 10 4. A new URSK is activated through Secure Ranging Setup Flow.

11 19.5.5 Flow Selection - Establish Secure Ranging

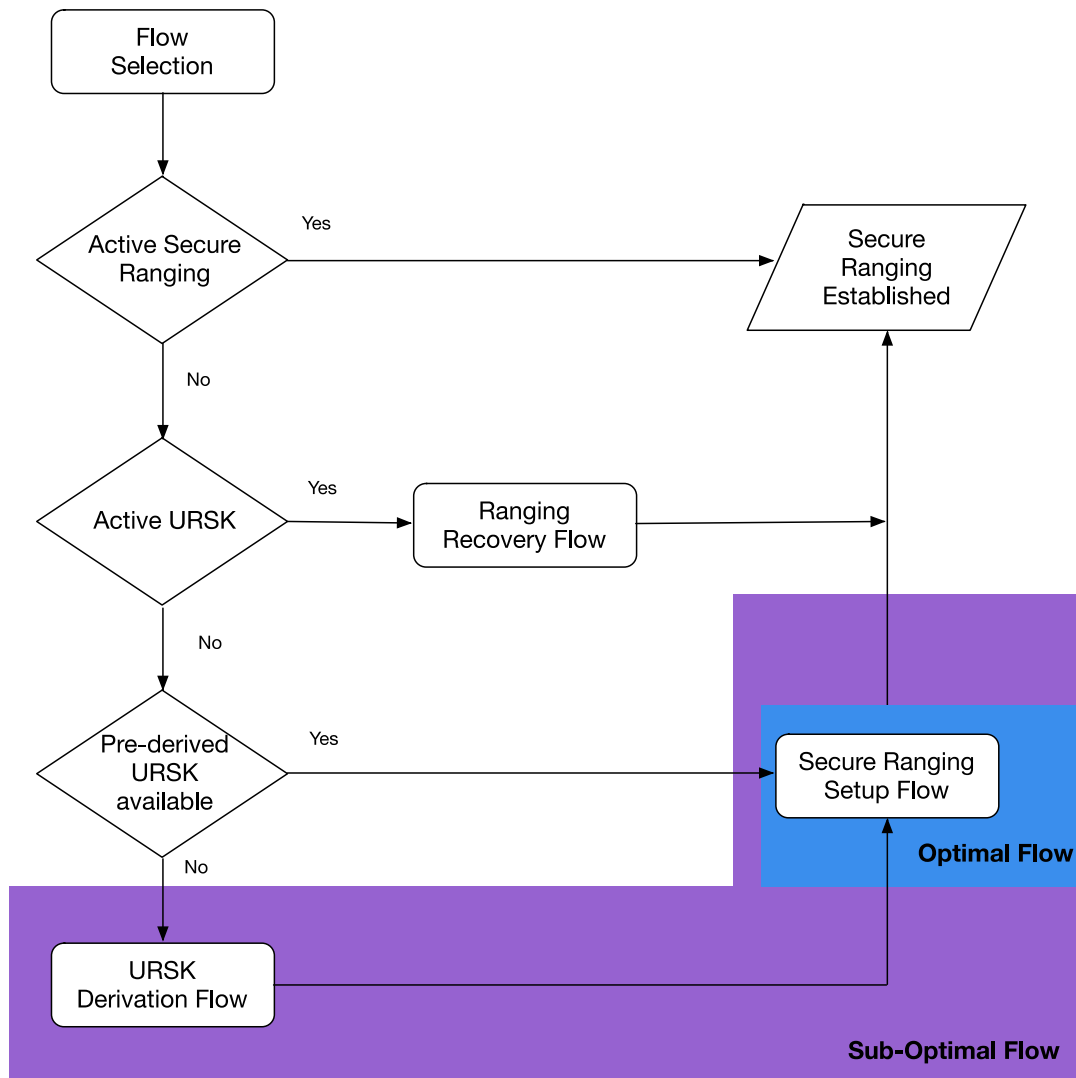
12 The vehicle has three options to establish secure ranging:

- 13 1. Optimal Flow
- 14 2. Sub-Optimal Flow
- 15 3. Ranging Recovery Flow

16 To decide which flow needs to be exercised for establishing secure ranging, vehicle shall go
17 through below flow selection. In the following, a ranging session is considered as an Active
18 Ranging Session when the vehicle and the device are exchanging UWB secure ranging messages
19 using the same URSK. This flow is shown in Figure 19-22.

1

Figure 19-22: Flow Selection for Establishing Secure Ranging



2

- 3 If the vehicle needs to localize the device, the vehicle shall first check which flow should be
- 4 initiated for establishing secure ranging. For this, the vehicle shall first check if it already has an
- 5 active ranging session. If it does, vehicle shall use that session to localize the device.
- 6 If there is no active ranging session, the vehicle shall then check whether an active URSK exists.
- 7 If it does, the vehicle shall follow the ranging recovery flow to recover the ranging session (using
- 8 its UWB_Session_Id) associated with that active URSK.
- 9 If there is no active URSK, the vehicle shall check whether a pre-derived URSK exist. If it does,
- 10 the vehicle shall follow optimal flow to establish secure ranging.
- 11 If there are no active or pre-derived URSKs, the vehicle shall fallback to sub-optimal flow.
- 12 If there is an active ranging session, and if the URSK TTL at the vehicle is about to expire or
- 13 vehicle decides to use a URSK with shorter TTL for engine start (see Section [19.5.6.1](#)), the
- 14 vehicle may setup a new secure ranging session with a pre-derived URSK. When the pre-derived
- 15 URSK becomes active, the vehicle and the device shall discard the previously active URSK (if

any). For more details on ranging recovery flow, optimal flow and sub-optimal flow, refer to below subsections.

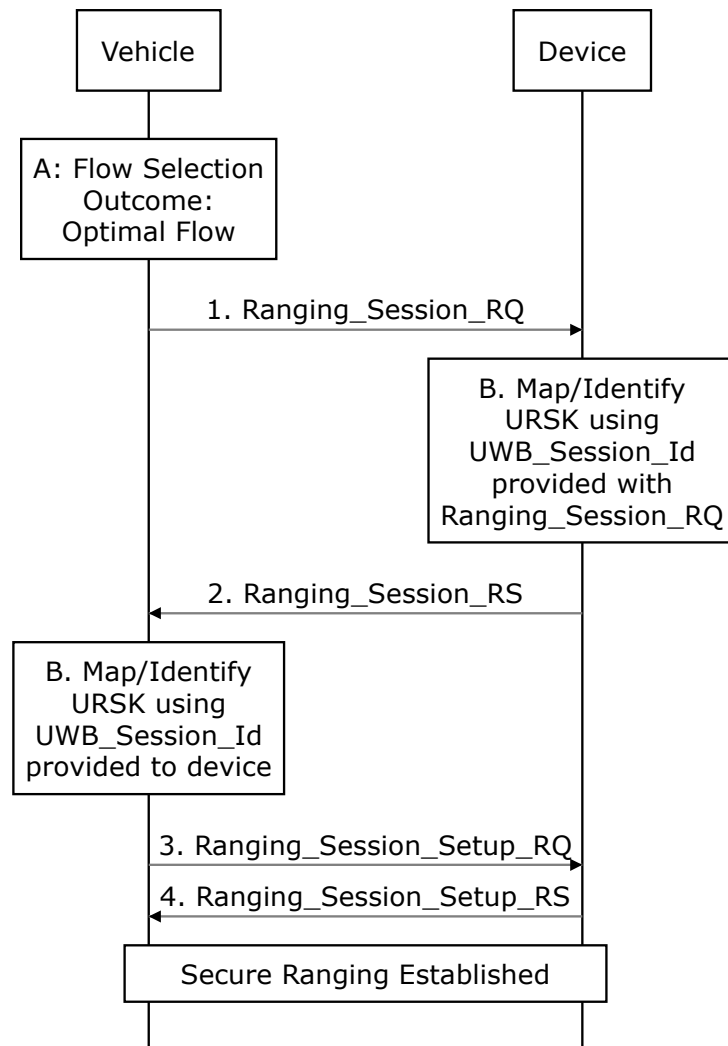
Optimal Flow

Optimal flow shall be exercised by vehicle, if it has a pre-derived URSK available. Optimal flow simply follows Secure Ranging Setup flow using a pre-derived URSK. If the vehicle uses this flow to activate a pre-derived URSK, the vehicle and device shall discard the currently active URSK (if any) for the associated Digital Key before the new URSK is activated.

To setup secure ranging, each message shall be encoded with Message Type: UWB Ranging Service message

The Secure Ranging Setup flow is shown in Figure 19-23.

Figure 19-23: Secure Ranging Setup Flow.

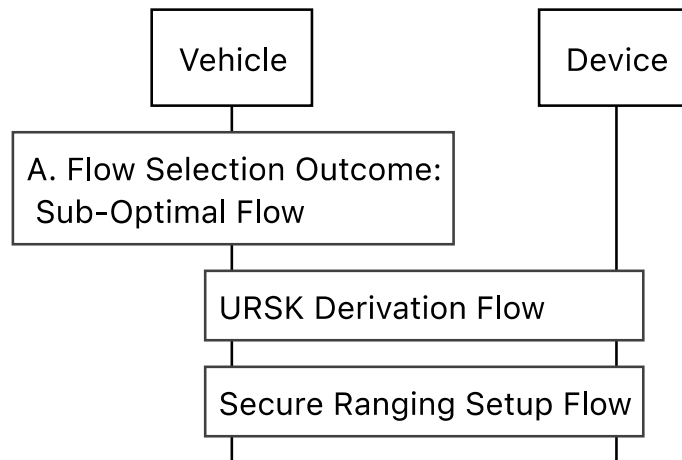


Sub-Optimal Flow

Sub-Optimal flow shall be exercised if a vehicle has no pre-derived URSK available. Sub-Optimal flow is a combination of URSK Derivation flow and Secure Ranging Setup flow.

Figure 19-24 illustrates the use of Sub-Optimal flow, in the absence of pre-derived URSK.

Figure 19-24: Sub-Optimal Flow.



In addition, if the device fails to activate an URSK as part of Optimal flow and no more pre-derived URSKs are available or when vehicle URSKs are out of sync with device URSKs (refer Section 19.8), vehicle shall fall back to Sub-Optimal flow.

Once secure ranging has been established by either using Optimal or Sub-Optimal flow, vehicle may localize the device. Once device localized, vehicle may perform any of the passive entry features/actions.

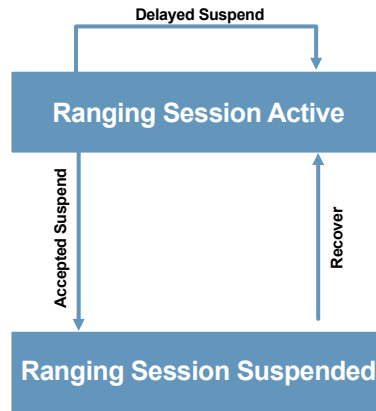
Ranging Recovery Flow

Once a ranging session has been established, it can be active for an extended period of time as long its URSK TTL has not expired or STS_Index has not reached max limit.

The ranging session shall be suspended if the BLE L2CAP connection has been disconnected, and if no encrypted BLE L2CAP connection is reestablished within 30 seconds of the initial BLE L2CAP disconnection.

However, as shown in Figure 19-25, even if the BLE connection is still active, either the vehicle or device may put an active ranging session in a suspended state for power optimization.

Figure 19-25: Ranging Session State Machine.



The decision on when to send a Ranging Suspend Request is up to the sender requester. However, the receiver may choose to delay the suspension request for a limited period of time. For example, device may request to put the ranging session to suspended state, if it does not detect any motion for extended period of time. On the other hand, vehicle may respond back with the request to delay the suspension, if it strongly believes ranging session is either still needed or will be in immediate future.

Figure 19-26 illustrates Ranging Suspend Accepted flow when requested by the device.

Figure 19-26: Ranging Suspend Accepted Flow.

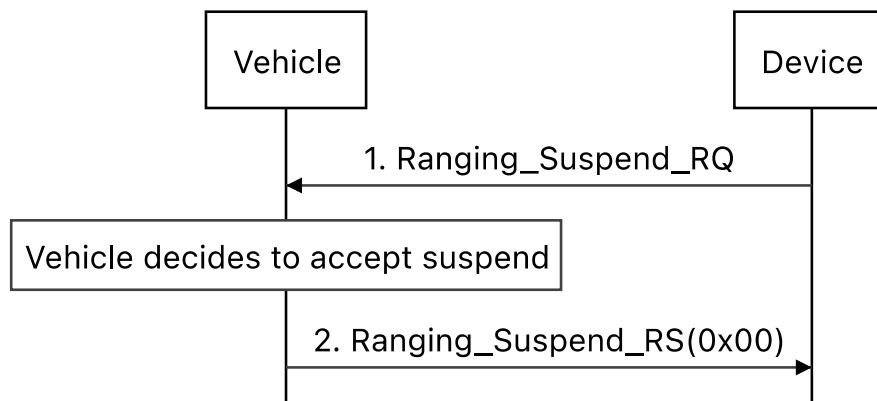
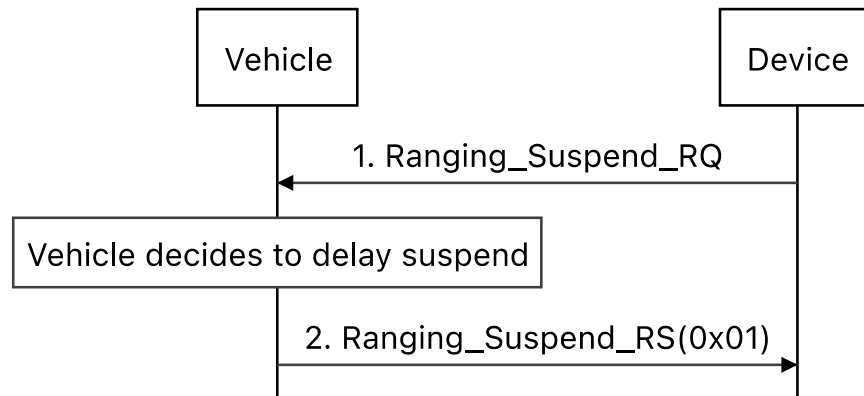


Figure 19-27 illustrates Ranging Suspend Delayed flow when requested by the device.

Figure 19-27: Ranging Suspend Delayed Flow.



When in suspended mode, vehicle may send a Ranging Recovery Request message (see Section 19.3.1.9) to the device. The device may trigger ranging recovery by sending Device Ranging Intent SubEvent to the vehicle

The Ranging Recovery flow offers a low latency based secure ranging which requires minimal exchange and no new URSK.

However, before initiating Ranging Recovery flow, vehicle shall meet the following requirements:

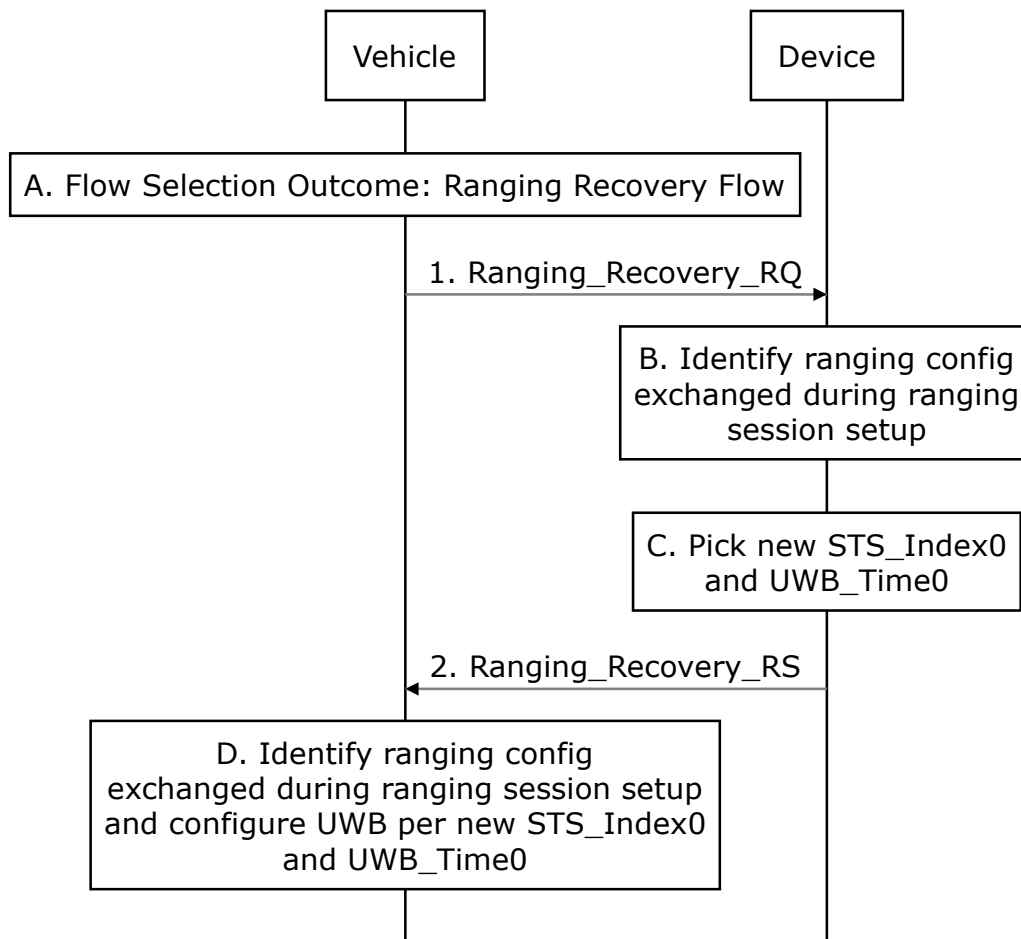
1. Has an active URSK (TTL has not expired) associated with the suspended ranging session in which the vehicle intends to recover
2. No change to ranging configurations of the suspended ranging session is needed, such as change of frequency, # of anchors, etc.

If vehicle needs a secure ranging, the vehicle shall check whether there is any suspended ranging session for the connected device that meets both of the above requirements. If it does, the vehicle shall initiate Ranging Recovery flow. When device receives the Ranging Recovery Request, it shall identify the same set of configurations used to establish the ranging session for the provided UWB_Session_Id. Device shall then pick and send a new UWB_Time0 and STS_Index0.

Figure 19-28 illustrates Ranging Recovery flow.

1

Figure 19-28: Ranging Recovery Flow.



2

3 For box A, refer to Section [19.5.5](#).

4 For box D, vehicle shall select previously negotiated UWB config parameters for
5 UWB_Session_Id associated with recovering ranging session.

6 Vehicle Low Power State

7 When the vehicle is in low power state, it may not start ranging after establishing a Bluetooth LE
8 connection. In such case the vehicle may send a vehicle state SubEvent notification indicating its
9 “Low power mode”. After receiving this notification, the device shall be responsible to start
10 ranging using the Ranging Intent SubEvent.

11 The device may immediately send the Ranging Intent or at a later point with higher confidence,
12 when a vehicle approach has been detected.

13 19.5.6 Standard Transaction over Bluetooth LE

14 If vehicle requires a standard transaction (e.g. in order to request an immobilizer token), the
15 standard transaction protocol as shown in Figure 7-1, followed by a Command Complete
16 SubEvent with Command_Status Deselect_SE, shall be performed.

Each APDU command and response shall be encoded as below.

Message Type: SE message

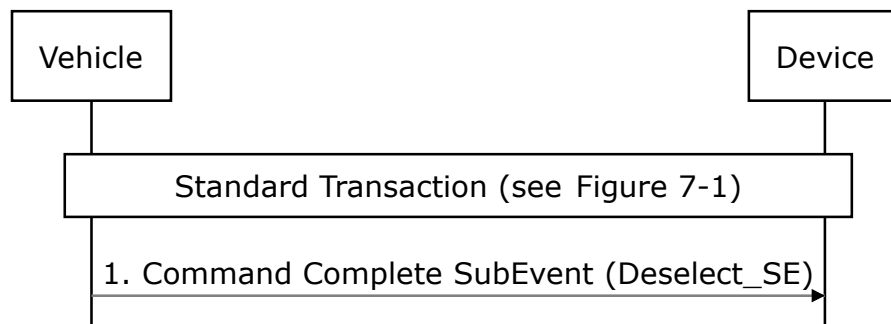
Message:

APDU Command Encapsulation: DK_APDU_RQ (see Section [19.3.2.1](#))

APDU Response Encapsulation: DK_APDU_RS (see Section [19.3.2.2](#))

Figure 19-29 illustrates the standard transaction flow.

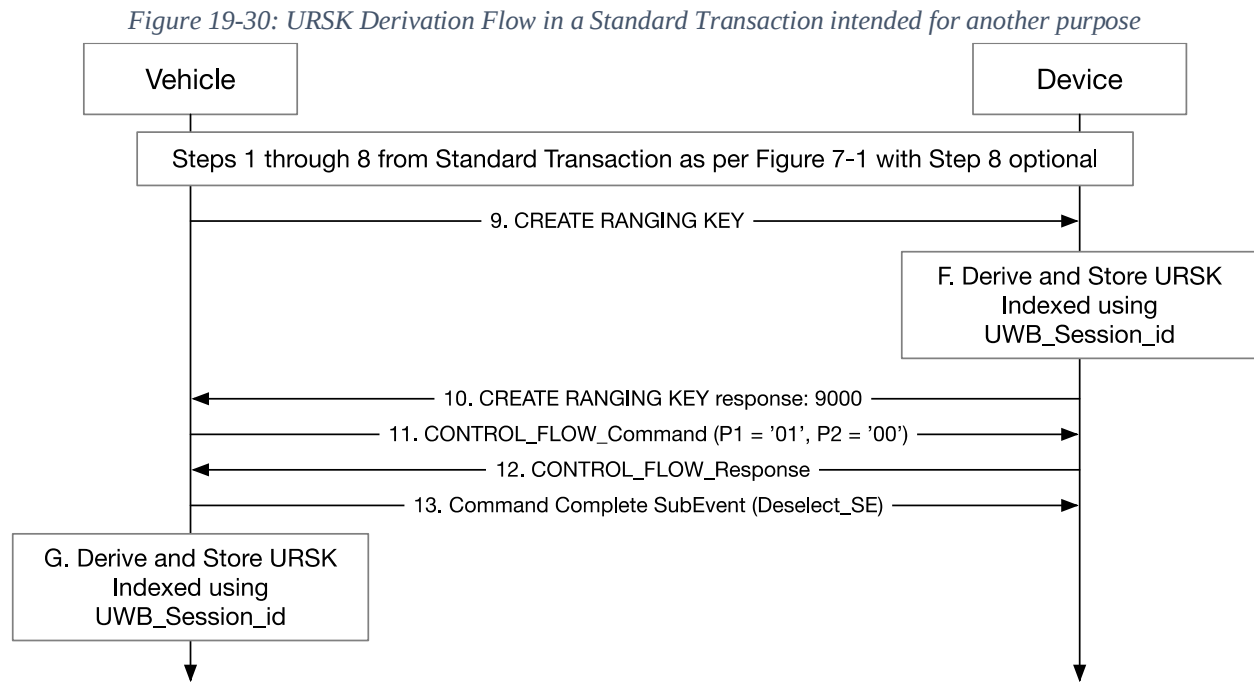
Figure 19-29: Standard transaction over Bluetooth LE.



19.5.6.1 Standard Transaction with URSK derivation via Bluetooth LE

Optionally, a URSK can also be derived within a standard transaction via Bluetooth LE that was mainly intended for other purposes like start engine, i. e., regardless of the AUTH0 Command Transaction Type Coding. The necessary steps for the URSK derivation are performed after an otherwise successful standard transaction with (optional) EXCHANGE commands. This is depicted in [Figure 19-30](#).

Note: Steps 1 through 8 in [Figure 19-30](#) are identical to Steps 1 through 8 from Figure 7-1.



If an error occurs within steps 1 to 8 of the standard transaction, URSK derivation is skipped and the appropriate error is signaled to the device. Vehicle then sends Command complete SubEvent (Deselect SE) to Device as per step 11 from Figure 19-30.

In case an error occurs during URSK derivation, only the action triggered with the preceding standard transaction is (already) accepted by the vehicle. If necessary, step 11: Command Complete SubEvent (Deselect SE) shall be sent to the device.

If a URSK derivation is still needed, a dedicated standard transaction for URSK derivation as described in section 19.5.4 should be performed.

19.5.7 Engine Start

Engine start shall be authorized for a limited time, only after completion of a standard transaction (which may also be used to retrieve an Immobilizer Token) and a UWB ranging.

The vehicle OEM may choose to setup a new ranging session using a different URSK with short TTL for engine start authorization. The usage of a short TTL (e.g. 30s) for the UWB ranging used for engine start increases overall security by reducing the general exposure of the URSK used for enabling engine start.

19.5.8 Receiver - Sharing & First Approach

19.5.8.1 Receiver Sharing

Digital Key solution supports sharing feature which allows vehicle owners to offer managed digital access of their vehicle with other receivers. For a Bluetooth LE/UWB capable vehicle, regardless of owner device's 'Wireless Capabilities', sharing ensures that a receiver device who may have UWB capabilities can enjoy Bluetooth LE/UWB based experience even when owner's device may not have Bluetooth LE/UWB capability.

To enable passive entry experience for receiver devices, vehicle provides its 'Wireless Capabilities' (7F49 template) and both device and vehicle derive Kble_oob_master as part of 'Owner Pairing' flow as described in Section 19.5.1.

During sharing, owner's device make use of Kble_oob_master and DK_Identifier to derive Kble_oob as described below.

Receiver Bluetooth LE Pairing: Receiver OOB Pairing Data Derivation

- $Kble_oob = HKDF\text{-}SHA256(16, Kble_oob_master, DK_Identifier)$
// (output_len, input_keying_material, info)

Where DK_Identifier is an 8-byte padded slot_identifier of the receiver. For example, if the slot_identifier is 6 bytes, it will have 0x0000 added as prefixed to make it 8 bytes.

Owner device shall send an updated 7F49 template (see Table 19-90) as part of the Import Request. This template is shared by the sender device to the receiver device as described in Section 11.4.2.

19.5.8.2 First New Key Transaction

For a receiver with NFC/Bluetooth LE/UWB capable device which has successfully received Digital Key access for owner's vehicle (Bluetooth LE/UWB capable), the receiver device can enjoy authorized RKE features using Bluetooth LE and passive entry feature without any further manual input or intent.

Note that the First New Key Transaction over Bluetooth LE is unrelated to any vehicle action such as lock or unlock. There is no requirement for user authentication for this transaction type.

After a successful Bluetooth LE Pairing Procedure, the vehicle shall initiate a First New Key Transaction. If the First New Key Transaction was successful, the vehicle shall end the Standard Transaction as defined in Figure 7-1 by a CONTROL FLOW command indicating First approach successful.

The Bluetooth LE Pairing Data shall be persisted:

1. For the vehicle, when the CONTROL FLOW command indicating First approach successful was sent,
2. For the device, when the CONTROL FLOW command indicating First approach successful was received.

Any error prior to this point shall lead to rolling back the Bluetooth LE pairing data. The device shall then retry the first approach flow on reconnecting to the vehicle.

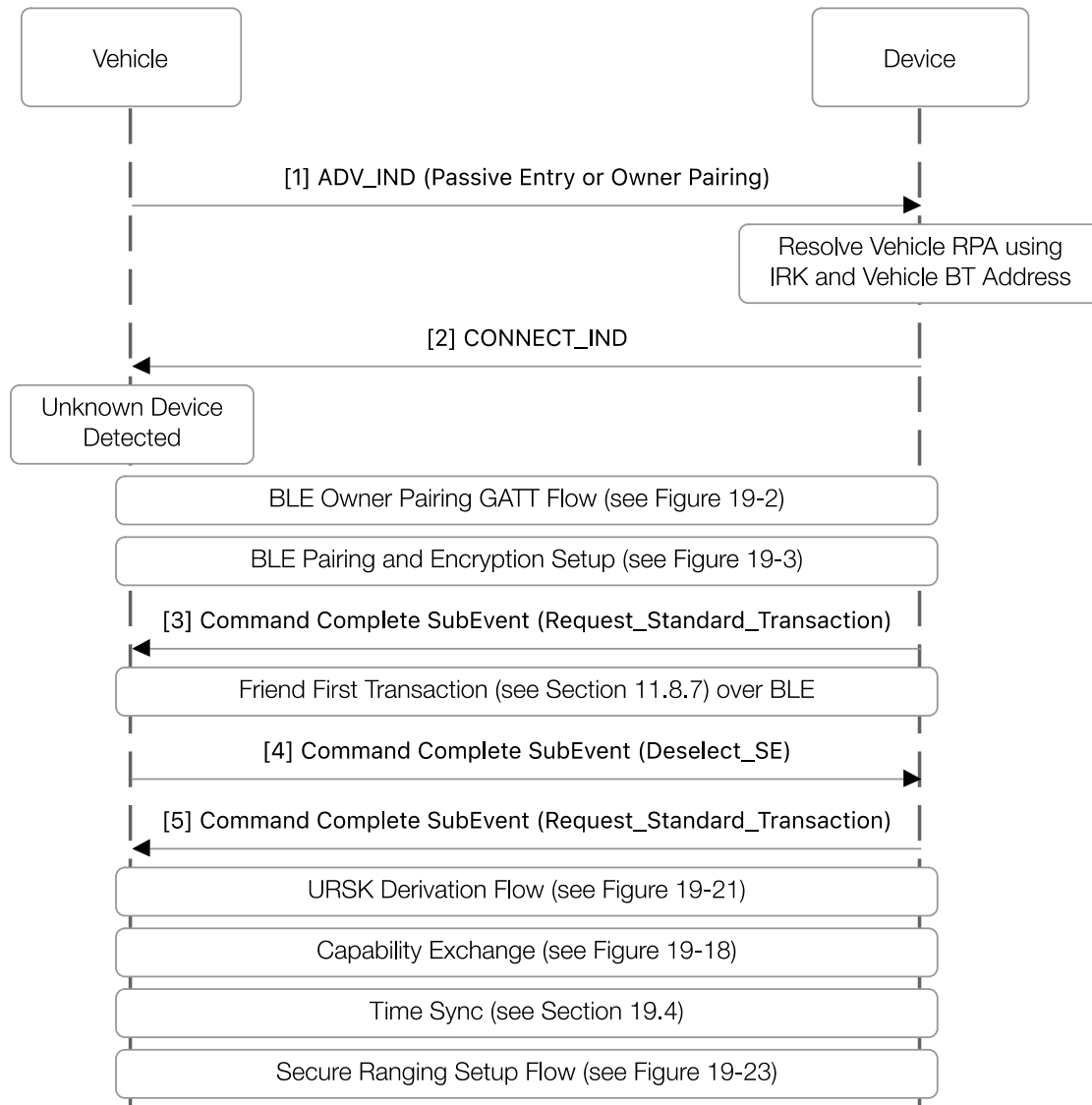
Figure 19-31 illustrates First New Key Transaction flow.

When performing friend first approach over BLE only, the following rules apply:

1. On completion of BLE Pairing and encryption setup, (Figure 19-3) the device shall send a Request_Standard_Transaction subevent within $t_{O_{veh-5}}$.
2. Vehicle and device shall complete friend first approach over the BLE link as outlined in Section 11.4.7.
3. Upon completion of friend first approach, the device shall send a Request_Standard_Transaction subevent within $t_{O_{veh-5}}$ to commence URSK derivation. If the vehicle does not receive a

Request_Standard_Transaction within timeout tO_{veh-5} , the vehicle shall initiate the Standard Transaction for URSK derivation.

Figure 19-31: First New Key Transaction Flow.



Bluetooth LE Link Layer Connection Establishment and L2CAP Setup Connection-Oriented Channel Setup

Upon receiving an ADV_IND from the vehicle, a receiver device shall first resolve the vehicle RPA using the IRK, and then shall send a CONNECT_IND to establish Bluetooth LE link layer connection. Once the Bluetooth LE link layer connection is established, the receiver device and vehicle shall establish L2CAP Connection-Oriented channel.

Bluetooth LE Pairing and Encryption Procedure

Both device and vehicle shall perform Bluetooth LE Secure OOB Pairing Prep flow, except for Kble_oob derivation (see Figure 19-16, except for box D).
Once vehicle has successfully derived Kble_oob, device shall initiate Bluetooth LE Pairing procedure by sending pairing request as described above in Section 19.2.1.4. Post successful pairing, device shall setup Bluetooth LE encryption before moving to First New Key Transaction and URSK Derivation flow as described in Sections 11.4.7 and 19.5.4, respectively.
Any Bluetooth LE messages being exchanged over-the-air as part of this flow shall not make use of DK message defined in Section 19.3.

Capability Exchange

For First New Key Transaction, the Capability Exchange shall be performed as described in Figure 19-17.

Time Sync

If the Device UWB Clock is “Not in sync” while Bluetooth LE connected with the device, the vehicle shall trigger the Bluetooth LE Timesync procedure 1 if all conditions are met (see Section 19.4.2)

19.5.9 RKE and RKE-Hands-Free Function Flow

19.5.9.1 User Authentication in RKE Function Flow

Remote transactions allow device to initiate on-demand features (e.g. lock, unlock and alarm) from within Bluetooth LE range. The protocol supports single event-triggered actions (e.g., locking on single tap) and enduring actions (e.g., holding a slider to adjust a window).
RKE Functions shall be supported by vehicles and devices with Wireless Capability Combinations WCC2 or WCC3 (see Table 19-79 for list of functions)
Before triggering a desired RKE action, the device shall perform user authentication (see Section 9). Before triggering a desired RKE-Hands-Free action, the device is not required to perform user authentication. If the user is successfully authenticated, the device shall request a challenge from the vehicle by sending RKE Request SubEvent. The device shall use the RKE Challenge and Execution ID received in RKE_AUTH_RQ, Function ID and Action ID corresponding to the Execution ID, and arbitrary data sent in RKE Request SubEvent (tag 87_h in 7F70_h), as arbitrary data to create a device signature using its private key for the requesting vehicle.
The device shall provide the user with the option to enable implicit UA, to enable explicit UA, or to disable use of RKE according to one of the allowed configurations defined in Table 19-92.
RKE policies may be selected individually for each Digital Key.

Table 19-92: RKE allowed user authentication configuration.

	Improved UX Policies	High Security Policies	
Configuration	Enable Implicit UA	Enable Explicit UA	Disable RKE
1	Supported	Supported	Supported
2	Supported	Supported	Not Supported

	Improved UX Policies	High Security Policies	
Configuration	Enable Implicit UA	Enable Explicit UA	Disable RKE
3	Supported	Not Supported	Supported
4	Not Supported	Supported	Supported
5	Not Supported	Supported	Not Supported

The implicit UA policy is described in Section [9.2](#).

The explicit UA policy shall require an additional UA to perform an RKE function (see Section [9.1](#)). This additional UA shall be performed after the intent to perform the RKE function has been registered and before the device signature for the RKE_AUTH_RS is generated for a specific key in the SE.

If usage of RKE is enabled, explicit UA should be selected by default.

To exchange RKE_Challenge and challenge response over BLE, it shall be encoded as below.

Message Type: Supplementary Service message

Message Type:

Message: RKE_Auth_RQ (see Section [19.3.5.1](#))

Message: RKE_Auth_RS (see Section [19.3.5.2](#))

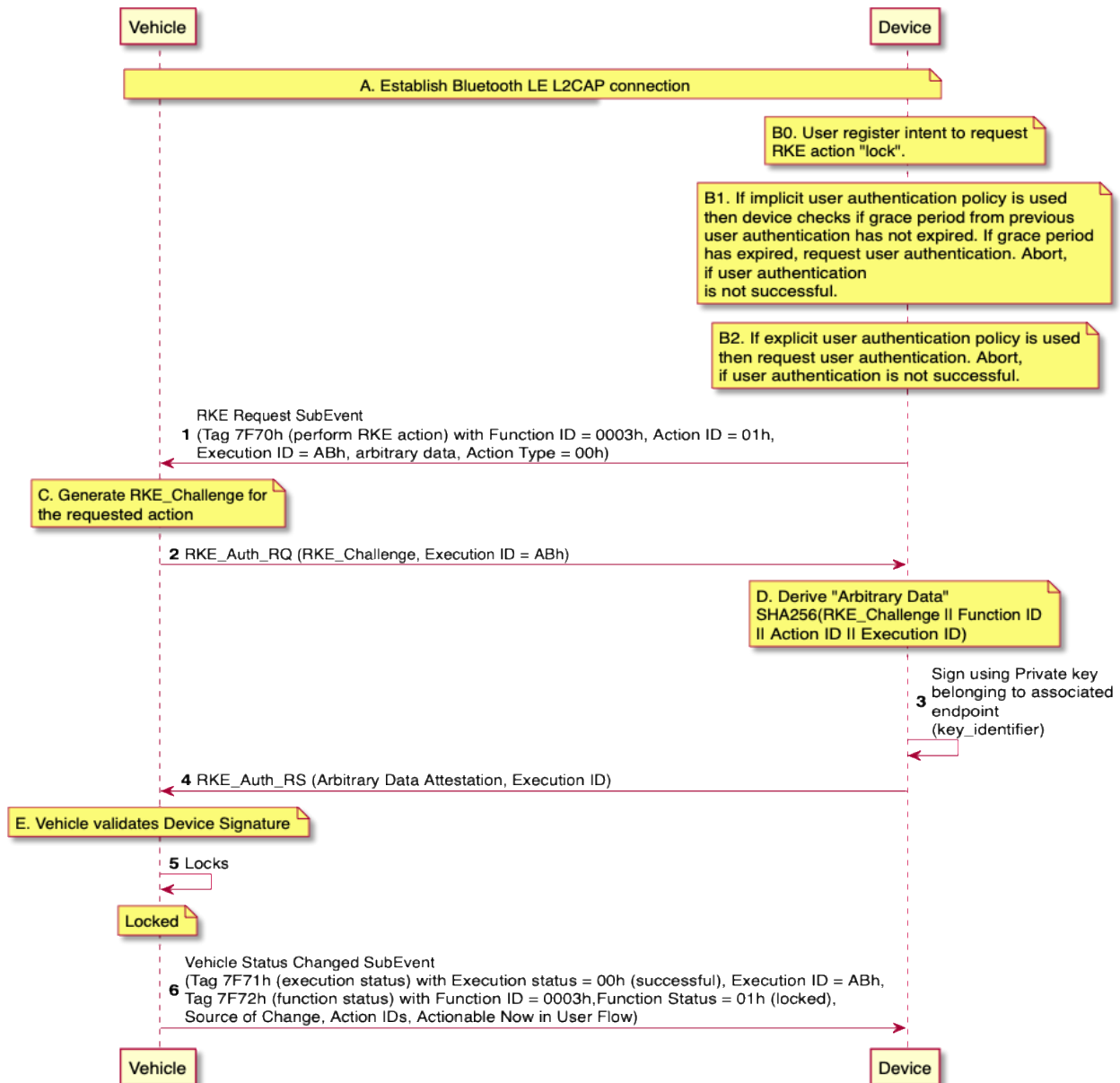
DK Event Notification (see Section [19.3.9](#))

19.5.9.2 Event-based RKE Action

Figure 19-32 provides an example flow for the event-based RKE flow with function status exchange.

1

Figure 19-32: RKE Flow for an Event-based RKE Action.



2

3 Box A operations are described in section 19.

4 In box B, the user triggers the action and performs user authentication (see Section 9).

5 In box D, device shall derive arbitrary data as described in 19.3.5.2.

6 In Box E, vehicle shall derive arbitrary data using the same procedure as the device and shall perform signature verification (see section 15), before action is executed.

7
8 In step 6, the successful execution together with a Vehicle function status change is
9 communicated via the Vehicle Status Changed SubEvent.

19.5.9.3 *Enduring RKE Action*

Enduring RKE allows an action to be performed as long as the user confirms that the action should be continued (e.g. by holding a slider to close a window). The vehicle may stop the function when an end state was reached, or an error occurred.

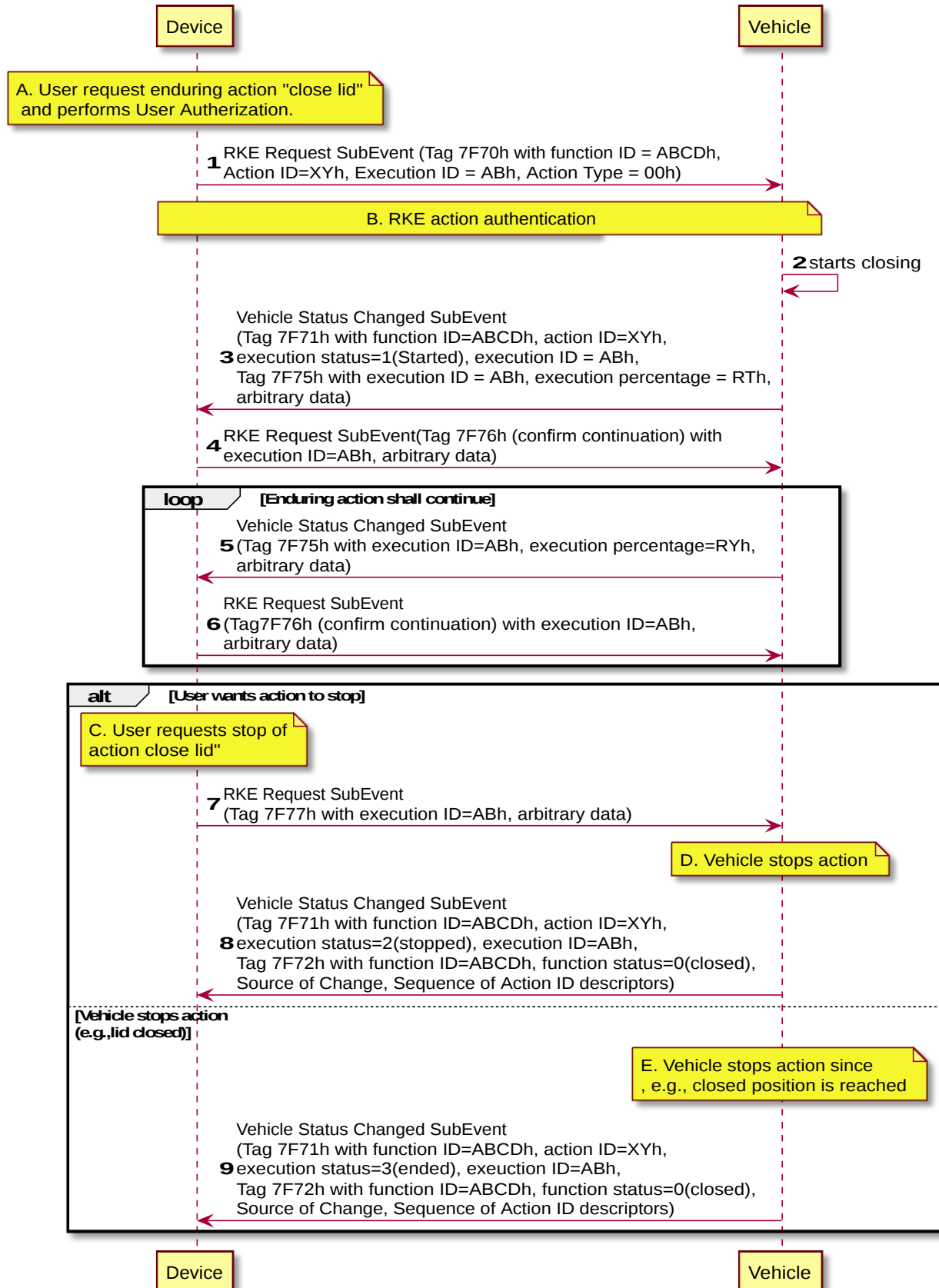
Enduring RKE offers two modes:

1. With Continuous Confirmation: (see loop with “Enduring action shall continue”) is required. In this case, the continuation confirmation shall only be sent if the user confirms the action on the user interface.
2. Without Continuous Confirmation: a certain action takes place until the device stops the execution (loop with continuation request is not present).

Figure 19-33 provides an example of an Enduring RKE action with Function id ABCD and action id XY are used for illustration only.

1

Figure 19-33: RKE Flow for an Enduring RKE action with continuous confirmation.



1 In box A, the user starts the Enduring RKE action.

2 After performing the RKE authentication as defined in Section 19.5.9.2, the Enduring RKE flow
3 starts. The confirmation interval is controlled by the vehicle. The arbitrary data as payload in the
4 Vehicle Status Changed SubEvent with “Request Continuation Confirmation” and, in the RKE
5 Request SubEvent with “Confirm Continuation” can be the basis of a vehicle OEM-specific
6 mechanism to ensure that the app on the device is still reachable or to perform round-trip time
7 measurements. If a “Request Continuation Confirmation” Vehicle Status Changed SubEvent
8 contains arbitrary data, the device shall return it unmodified in the corresponding “Confirm
9 Continuation” RKE Request SubEvent. This should happen within the next connection event
10 (30ms recommended). It is up to the Device OEM implementation to call the callback before or
11 after sending the “Confirm Continuation” RKE Request SubEvent.

12 While the user is actively using an Enduring RKE function with continuous confirmation on the
13 device for the given vehicle, and if the vehicle does not receive a continuation confirmation
14 within 100ms after a continuation confirmation request was sent, it may stop the function. This
15 time measurement is between the over-the-air packet sent by vehicle and the over-the-air packet
16 received by the vehicle.

17 In box C, the user wants to stop the Enduring RKE action. This is indicated by the device via the
18 “stop enduring action” payload.

19 In box E, the vehicle initiates the stop of the Enduring RKE action. The device can be notified
20 when an enduring function reaches a certain stop state or when an error occurs.

21 The Vehicle Status Changed SubEvent may contain one or more function status updates along
22 with the Continuation Confirmation Request.

23 *19.5.9.4 Vehicle Function Status Update*

24 Figure 19-34 shows the possible flows for requesting a vehicle function’s status and being
25 informed about status changes. The subscription mechanism allows the device to subscribe to
26 vehicle OEM-specific function IDs, the vehicle shall send function status updates when the status
27 of those function IDs changes. With each subscription message, the vehicle shall delete the
28 current subscription ranges and take over the new ranges. To unsubscribe from updates, a single
29 function range with “from function id” = 0 and “to function id” = 0 shall be sent.

30 Besides subscription, the status of all supported functions or the status of specific functions can
31 be requested. When all supported functions are requested by the device, the vehicle shall only
32 return the function IDs with status that are supported by the vehicle. The function status report
33 can be split in multiple messages depending on the maximum supported L2CAP SDU size.

34 No subscription is necessary for standard functions. Upon connection establishment, after owner
35 pairing, and following Friend First Approach, the vehicle shall send Supported Function and
36 Action Summary (Tag 7F79_h) to this device. Upon connection establishment, after owner pairing,
37 and following Friend First Approach, the vehicle shall send status update (Tag 7F72_h) to this device.
38 Whenever the vehicle status changes or Actionable Now in User Flow property of an Action ID
39 alters, the vehicle shall send status updates (Tag 7F72_h) to all connected devices.

40 Source of Change (Tag 96_h) shall be set to ‘unknown’ in the Tag 7F72_h in the initial vehicle status
41 update message. Source of Change (Tag 96_h) shall be set to appropriate value in the Tag 7F72_h
42 when vehicle status changes, and the vehicle is aware of the source that caused the function status

1 change. Source of Change (Tag 96_h) shall be set to ‘Not Applicable in the Tag 7F72_h when
2 Actionable Now in User Flow property of an Action ID alters but the function status is unchanged.

3 The explicit status request shall only be executed on explicit user intent or for device error
4 recovery. Afterward, changes shall be tracked via the function status updates send out on status
5 change.

6 When entering low power mode, the vehicle shall send Vehicle Status Changed SubEvent
7 indicating it is in low power mode to all connected devices. If the device has received a Vehicle
8 Status Changed SubEvent indicating the vehicle is in a low power mode, the device shall not
9 send a RKE Request SubEvent for status request updates except for device error recovery. After
10 exiting lower power mode, the vehicle shall send a status update for all connected devices.

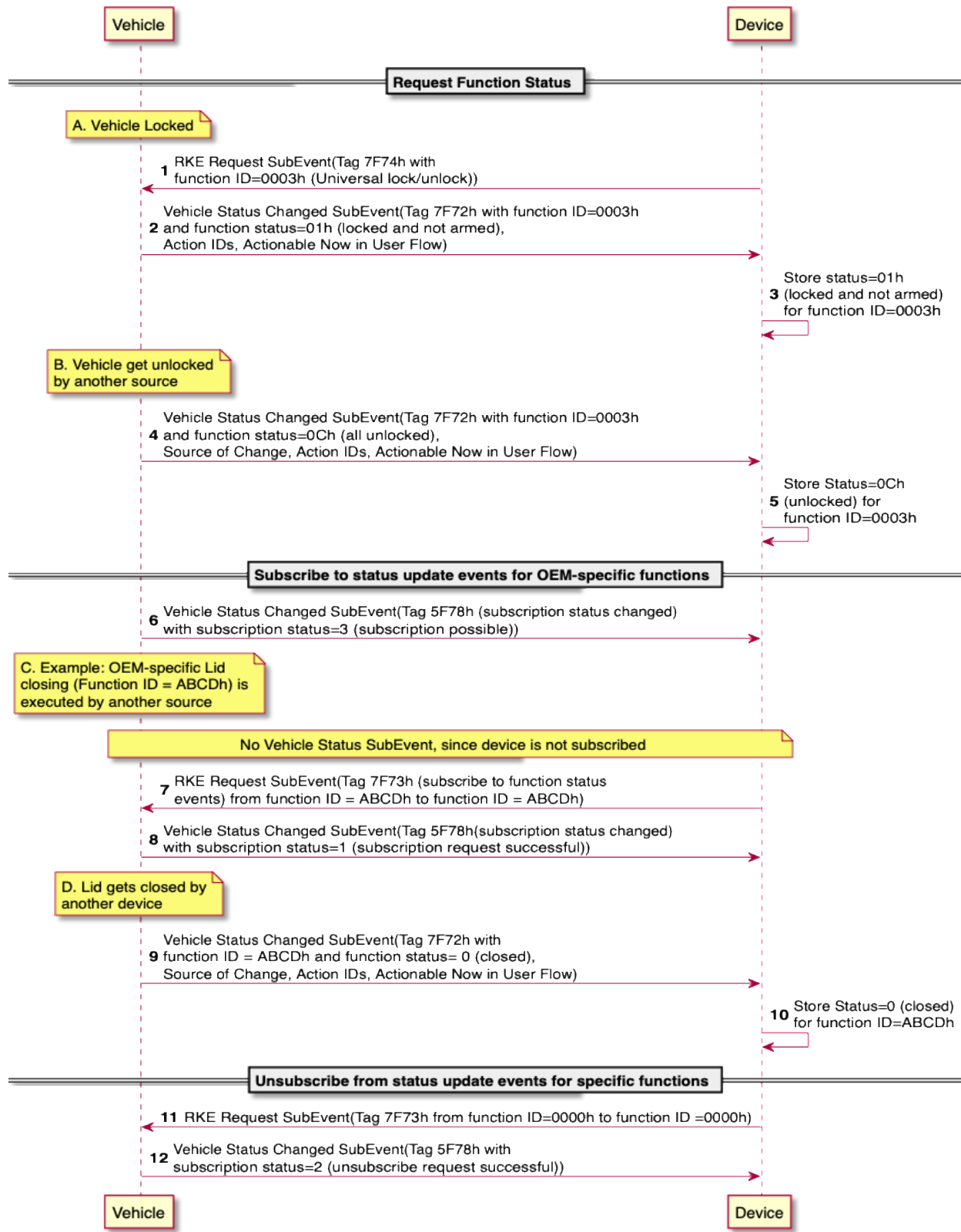
11 If the vehicle is in low power mode, the status request shall not be sent except for device error
12 recovery. After establishing the Bluetooth LE connection, the device may subscribe to status
13 updates, only when the vehicle indicates first the availability of the subscription mechanism (see
14 message in step 6 of Figure 19-34). The vehicle shall indicate when it stops the automatic status
15 updates for the subscribed ranges (e.g., due to entering low power mode) and when a
16 resubscription is possible again.

17 When a function is not supported by the vehicle, the vehicle shall only send a status update
18 informing about the unavailability (cf. Table 19-82, status F0_h), when the status is explicitly
19 requested by the device (“get function status”).

20 If the Bluetooth LE connection between the device and the vehicle is available and the
21 subscription to a certain function id range is active, the framework shall store the latest received
22 function status values for this range of function ids. The device shall provide an API for eligible
23 vehicle OEM apps for subscribing to a certain function id range, requesting a certain function
24 status value from the vehicle and reading the stored status data.

1

Figure 19-34: Vehicle Function Status Retrieval and Event-based Update.



2

19.5.9.5 Device Ranging Intent SubEvent Feedback

The vehicle shall send Tag 7F78_h in Vehicle Status Changed SubEvent after receiving Device Ranging Intent SubEvent from the device and after localizing the device with UWB ranging.

The device can use Device Ranging Intent SubEvent feedback for improving triggering of Device Ranging Intent SubEvent. The Device Ranging Intent triggering mechanism and use of Device Ranging Intent SubEvent feedback in it is out of scope of this specification.

19.5.9.6 RKE-Hands-Free Flow

With RKE-Hands-Free, users indicate to the device their intent to perform an action for a function on the vehicle, without requiring user authentication.

The user can input intent to perform an action for a function using gestures, voice, etc., on the device. The device maps the user intent to perform an action for a function to one of the Action IDs for a Function ID. How the intent is input onto the device for RKE-Hands-Free and how the intent is mapped to a function/action is out of scope of this specification.

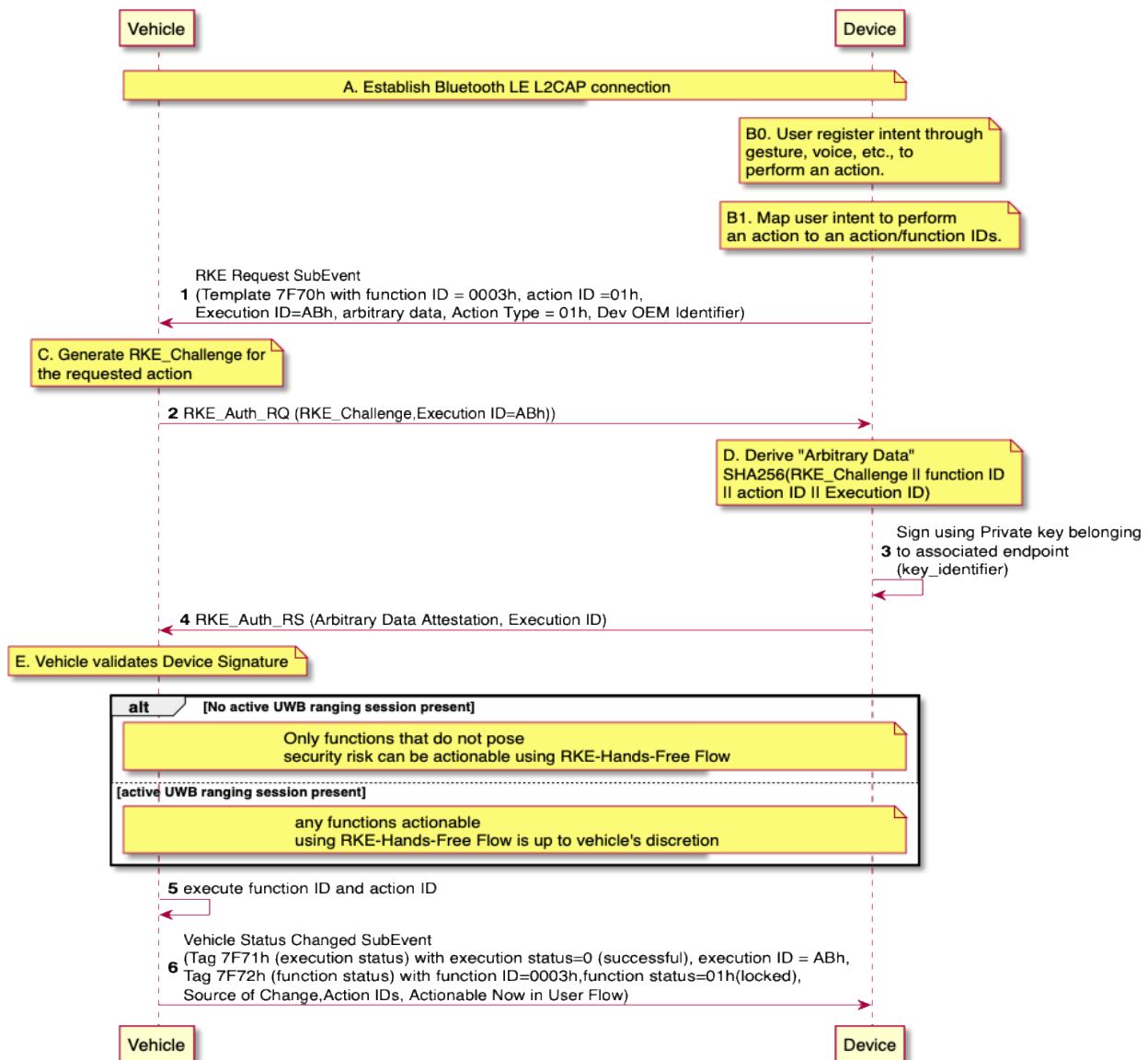
The support of RKE-Hands-Free is optional for the device and the vehicle. RKE-Hands-Free actions can be supported by vehicles and devices with Wireless Capability Combinations WCC3.

The key steps in RKE-Hands-Free operation are enumerated below.

1. User Initiation: User input RKE-Hands-Free action onto device. The RKE-Hands-Free action is converted to RKE-Hands-Free command sent to the vehicle over encrypted Bluetooth link.
2. Car Validation: Upon receiving the RKE-Hands-Free action, the vehicle localizes the device with UWB secure ranging. This ensures that the user's device is physically present nearby.
3. Action Execution: If the vehicle successfully localizes the user's device within proximity, it proceeds with the requested action. In such situations, it can execute any action including passive-entry related actions, such as unlocking.
4. Security measures in case of non-localization: If the vehicle cannot localize the user's device (e.g., too far for UWB ranging to operate), it remains vigilant. In such situations, vehicle shall not execute passive entry-related actions but can execute actions that it deems not security sensitive. Any action that pose a security risk shall be disallowed to prevent unauthorized access in this case.

1

Figure 19-35: RKE-Hands-Free Flow for an Event-based Action



2

3 Figure 19-35 illustrates an example of RKE-Hands-Free, showing the sequence of steps
4 involved, which are described below.

5 Box A operations are outlined in section 19. Upon connection establishment, the vehicle sends to
6 the device, list of all the functions/actions supporting RKE-Hands-Free and the actionable
7 functions/actions that are currently actionable using RKE-Hands-Free (see section 19.5.9.4). In
8 box B0, the user inputs their intent to perform an action for a function. In box B1, the device
9 maps the user input to one of the actions for a function in the Table 19-81.

10 The device shall then look up the input function/action in the list of all Function IDs that support
11 RKE-Hands-Free flow. If no match is found, it can notify the user that the input function/action
12 is not supported in RKE-Hands-Free flow.

13 In box E, the vehicle shall derive arbitrary attestation data using the same procedure as the
14 device and shall perform signature verification. The vehicle shall ensure signature is verified

before executing the corresponding Function ID and Action ID. The vehicle shall ignore Usage value (Tag 93_h in Table 15-58) in Arbitrary Data Attestation in RKE-Hands-Free. To prevent unauthorized access, the vehicle shall rely on UWB secure ranging to localize the device before executing RKE-Hands-Free actions.

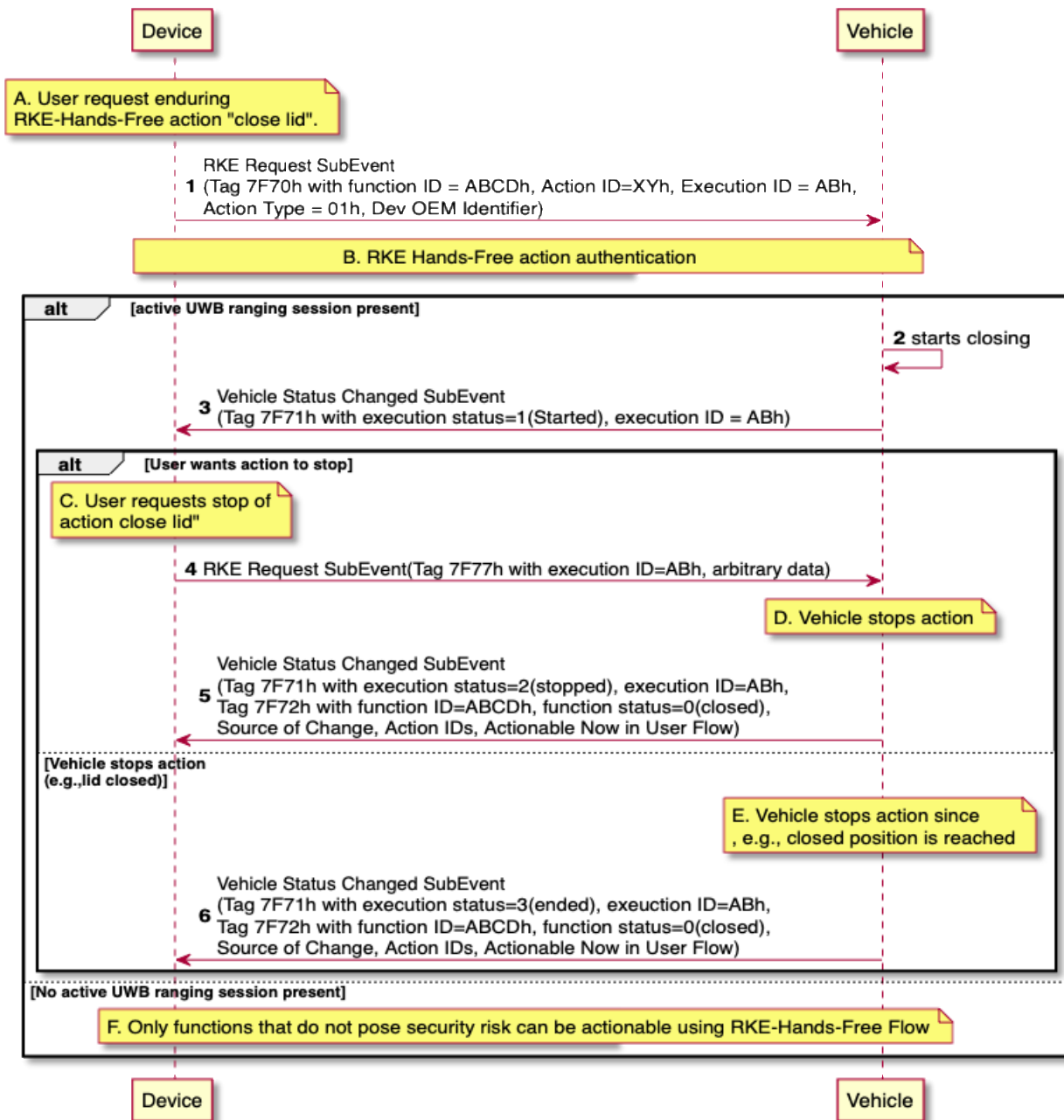
If there is no active UWB ranging session between the device and the vehicle, the vehicle shall allow only functions/actions not posing a security risk to be actionable with RKE-Hands-Free flow. Passive entry-related functions/actions shall be disallowed in this case. In the presence of an active UWB ranging session, however, the vehicle decides whether to allow any function/action to be actionable with RKE-Hands-Free flow.

After executing the function, the vehicle sends execution status (7F71_h) including the Execution Status (82_h) and the function status (7F72_h) indicating the outcome of the received request.

19.5.9.7 Enduring RKE-Hands-Free Flow

This specification does not define RKE-Hands-Free enduring with continuous confirmation. The reason is that true continuous confirmation would necessitate recursive user input to perpetuate execution, which runs contrary to the nature of hands-free operation (see section 19.5.9.3). Figure 19-36 shows RKE-hands-Free Flow for an Enduring RKE-Hands-Free action without continuous confirmation.

1 *Figure 19-36: RKE-Hands-Free Flow for an Enduring RKE-Hands-Free action with confirmation*



2

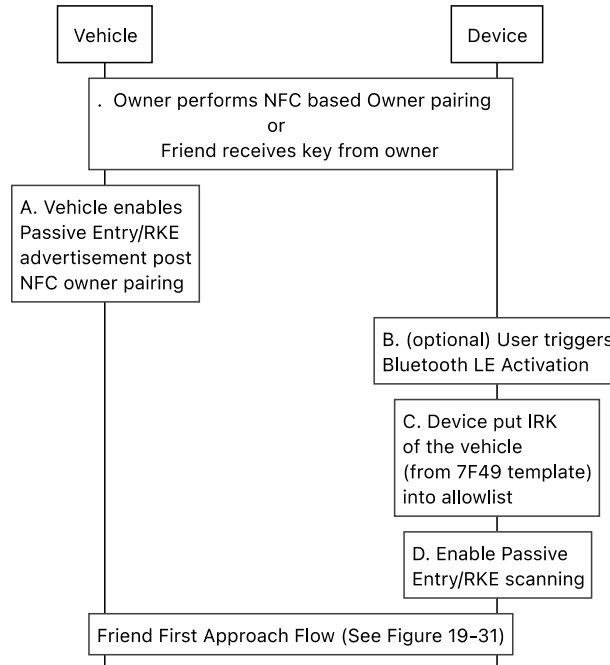
3 **19.5.10 Bluetooth LE Activation Flow**

4 If a user goes through NFC based owner pairing or First New Key Transaction even though both
 5 device and vehicle support Bluetooth LE/UWB capability, DK solution allows the user to
 6 migrate NFC provisioned key to Bluetooth LE/UWB based passive entry experience without
 7 user having to perform owner pairing or key sharing again.

8 If a user goes through NFC based owner pairing or First New Key Transaction for a DK system
 9 which supports Bluetooth LE without UWB, DK solution allows the user to migrate NFC

provisioned key to Bluetooth LE based RKE experience without user having to perform owner pairing or key sharing again
This NFC to Bluetooth LE migration may be triggered by user anytime even when the vehicle is not within the Bluetooth LE proximity. Upon first approach after user triggered migration, device and vehicle shall establish Bluetooth LE bonding and offer passive entry or RKE experience for that approach.
Figure 19-37 illustrates Bluetooth LE Activation.

Figure 19-37: First Approach Activation.



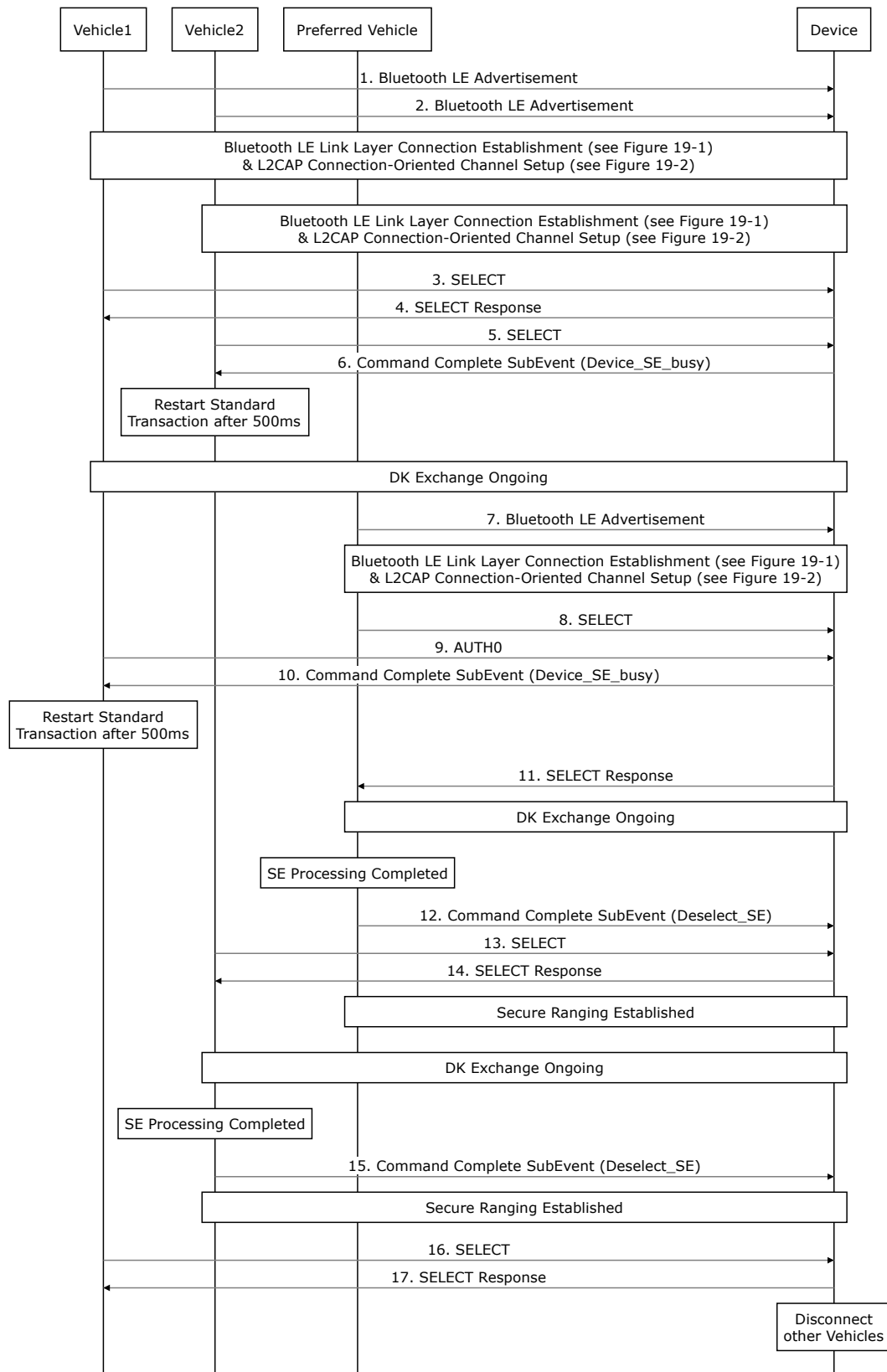
19.6 Digital Key - Preference Management

19.6.1 Preference Management: Connected Vehicles < Connection Limit

A user with multiple Digital Keys can mark any one of the Digital Keys as preferred or default. When such user approaches multiple valid vehicles, the device shall prioritize the SE access for vehicle with the default Digital Key and put the other vehicle's SE request on hold by sending 'Device_SE_busy', if they compete for resources with the preferred Digital Key. When vehicle receives 'Device_SE_busy' SubEvent, it shall wait for a defined wait time (e.g. two seconds) before restarting the standard transaction again. Once the device has completed SE processing with the preferred vehicle, it shall continue with the rest of the vehicles. Figure 19-38 provides an example flow for device preference management.

1

Figure 19-38: Device Preference Management.



2

19.6.2 Preference Management: Connected Vehicles $\geq$ Connection Limit

When a device is already connected to its max limit Bluetooth LE connections and user approaches (within Bluetooth LE range) additional vehicle for which it has a valid Digital Key with the intent to access it, user may go through one of the following two scenarios:

1. New Vehicle is Preferred
2. New Vehicle is Non-Preferred

For scenario #1, when the device detects advertisement from a new vehicle which is configured as preferred vehicle for the device, it may drop one of the connected vehicles to prioritize the Bluetooth LE connection for the preferred vehicle.

For scenario #2, when the device detects advertisement from a new non-preferred vehicle, it shall drop one of the non-preferred connected vehicles (which connection to drop is device OEM's implementation) to prioritize the Bluetooth LE connection for the new vehicle

When there are more vehicles than device's max Bluetooth LE connection limit, user shall be able to exercise passive entry experience with any of the connected vehicle. However, for non-connected, non-preferred vehicle, user may have to fallback to RKE feature.

To exercise RKE feature in this scenario, user selects the right vehicle and requests RKE feature. This shall prioritize the non-connected vehicle for Bluetooth LE connection for user to gain access. Once connected and unlocked, vehicle should try to establish secure ranging with newly connected device to move the user to passive entry experience and for device in/out detection for engine start

19.7 SubEvent Handling

19.7.1 Command Complete SubEvent Code Handling

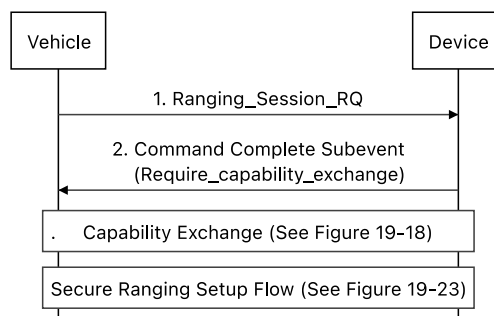
This section defines how vehicle shall recover from a SubEvent defined in Section [19.3.9](#).

19.7.1.1 Require Capability Exchange

Ranging capabilities are first negotiated between device and vehicle as part of the owner pairing flow (See Section [19.5.1](#)) using Ranging_Capability_RQ and Ranging_Capability_RS.

Figure 19-39 provides an example flow for handling Require Capability Exchange SubEvent.

Figure 19-39: Required Capability Exchanged.



1 19.7.2 Ranging Session Status Changed SubEvent

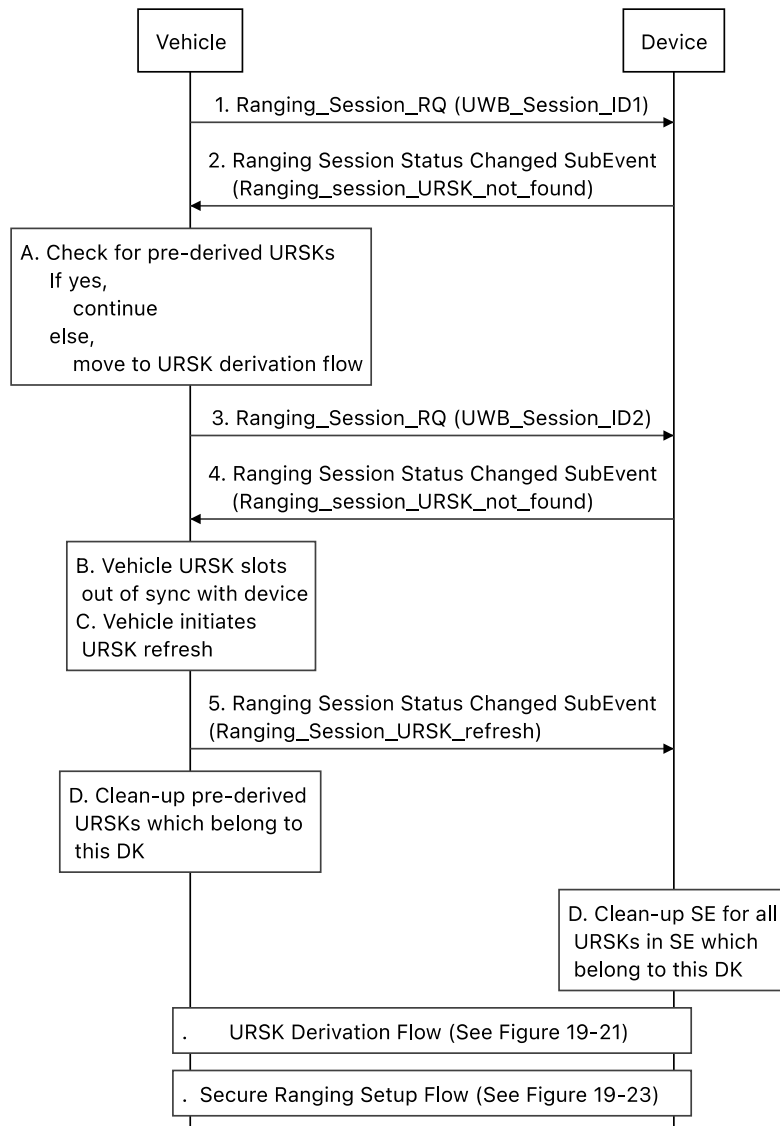
2 19.7.2.1 URSK Not Found

3 When a device fails to retrieve the associated URSK for a requested UWB_Session_Id, it shall
4 respond with a 'Ranging Session Status Changed SubEvent' with parameter value
5 "URSK_not_found".

6 The vehicle should clean up the pre-derived URSKs on the device if it fails to retrieve URSKs
7 repeatedly.

8 Figure 19-40 provides an example flow for URSK Retrieval Failure handling.

9 *Figure 19-40: URSK Not Found.*



10

11 For box A, vehicle shall query for stored URSKs.

12 For box B, it is vehicle's decision on when to initiate URSK Refresh flow.

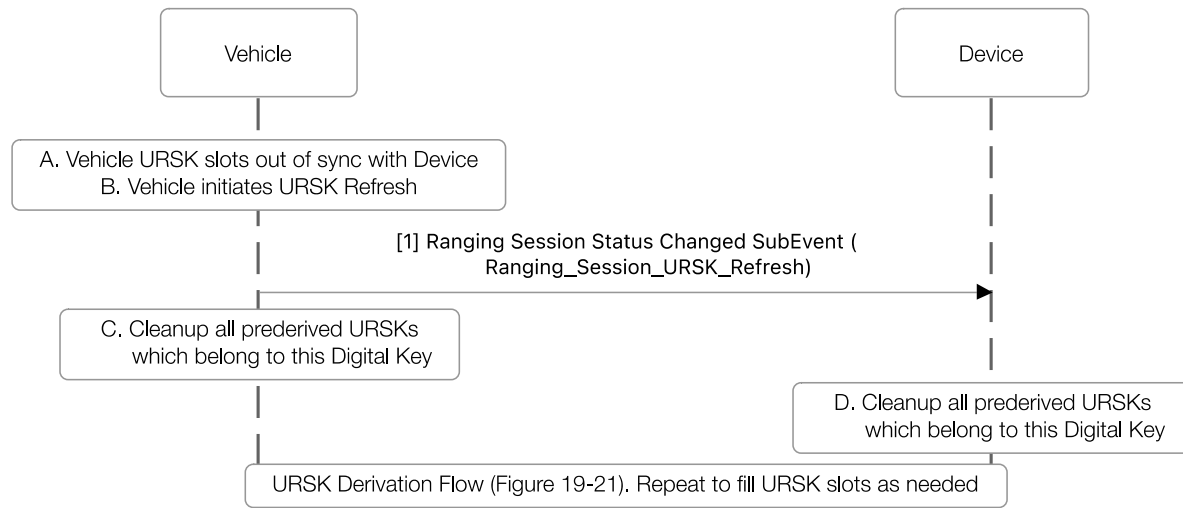
13 For box D, vehicle should delete URSKs associated with the digital key.

19.7.2.2 URSK Refresh

The vehicle may trigger a deletion of all pre-derived URSKs in the device's SE at any time by sending a Ranging Session Status Changed SubEvent Notification with Session_Status "Ranging_session_URSK_refresh". This method shall not affect the URSK used in the active ranging session.

Figure 19-41 provides an example flow for URSK Refresh flow.

Figure 19-41 URSK Refresh Flow

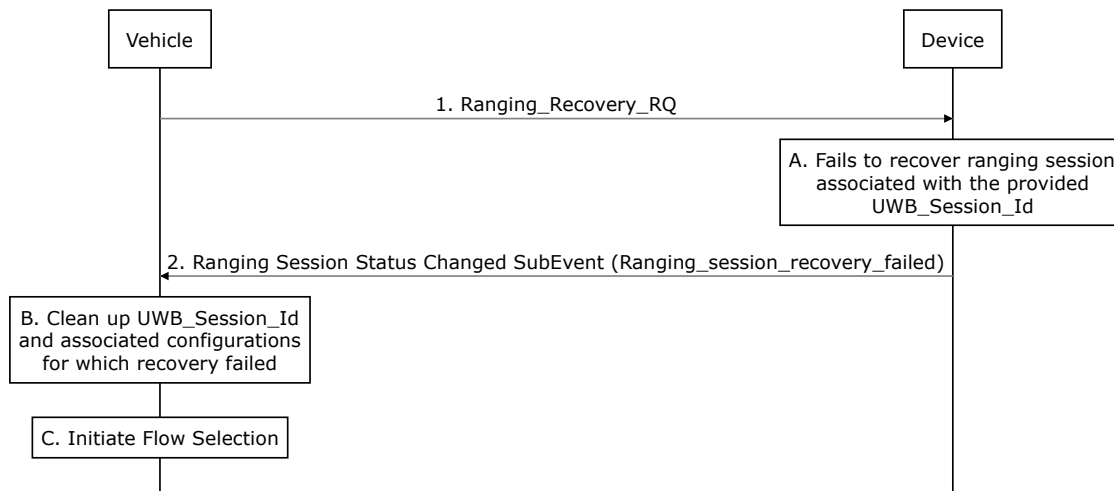


19.7.2.3 Recovery Failed

When the device fails to recover a ranging session, which was requested by vehicle, the device shall respond to the vehicle with Ranging Session Status Changed SubEvent Notification with Session_Status "Ranging_session_recovery_failed". Upon receiving this notification, the vehicle shall discard the corresponding URSK. A new ranging session should then be established according to the flow selection (see Figure 19-22).

Figure 19-42 provides an example flow for "Recovery Failed" handling.

Figure 19-42: Recovery Failed Flow.



For box B, process the received message, the vehicle removes the URSK, configurations and any other metadata associated with UWB_Session_Id for which recovery failed.

For box C, refer to Section [19.5.5](#).

19.8 Error Handling

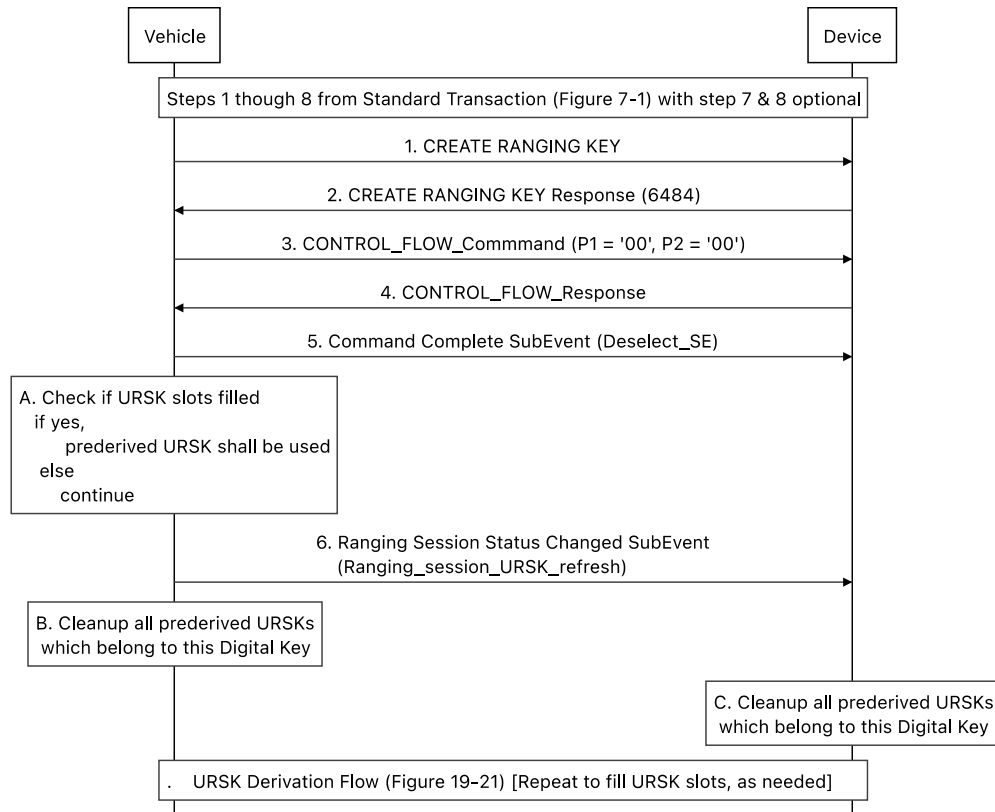
19.8.1 SE Transaction Recovery

19.8.1.1 CREATE RANGING KEY Failure

If device URSK slots for a given Digital Key are filled, it shall fail to store any new key. In this event, the device shall respond to CREATE RANGING KEY command with Status Word 6484_h, which means ‘URSK slots have been exhausted’. Upon receiving status word 6484_h, vehicle shall not request new URSK derivation unless it requests “URSK_refresh” as outlined in Figure 19-43 first.

If vehicle identifies that device and vehicle are out of sync with pre-derived URSKs, vehicle may go through URSK Refresh flow (see Figure 19-41). Figure 19-43 provides an example flow for Create Ranging Key Failure.

1 *Figure 19-43: URSK Refresh due to Status Word 6484_h in CREATE RANGING KEY Response.*



2

3 For box A, vehicle shall query for pre-derived URSKs.

4 For box B, vehicle shall delete all pre-derived URSKs associated to the Digital Key.

5 For box C, device shall delete all pre-derived URSKs associated to the Digital Key.

6 *19.8.1.2 Others*

7 Error handling for rest of the applet commands shall be done based on the Status Words and
8 flows defined in Section [15](#).

20 UWB MAC AND CHANNEL ACCESS [WCC3]

This section defines the MAC layer and channel access protocols for UWB ranging for Digital Key.

20.1 UWB MAC Architecture

20.1.1 Overview

This subsection describes the concepts that are used in the UWB MAC (see [31] and [33]).

20.1.1.1 Ranging Roles Definition

In the DK UWB ranging service, the ranging device roles are based on which device starts the ranging procedure and is responsible for setting the ranging exchange. The following role definitions apply to UWB layer only:

1. An entity that starts the UWB ranging packet exchange by sending a first UWB POLL packet is called an “initiator.” In the case of DK this is the device.
2. An entity that responds to a UWB POLL packet is called a “responder”. In the case of DK, these are the anchors on the vehicle.
3. An entity that includes a number of responders is called a “responder-device”. In the case of DK, this is the vehicle.
4. An entity that controls the ranging procedure by sending a Pre-POLL packet is called the controller. In the case of DK, this is the device, i.e. the device is the initiator and the controller

Note that the roles definition above may not apply to the Bluetooth LE layer.

20.1.1.2 Logical and Physical Responders

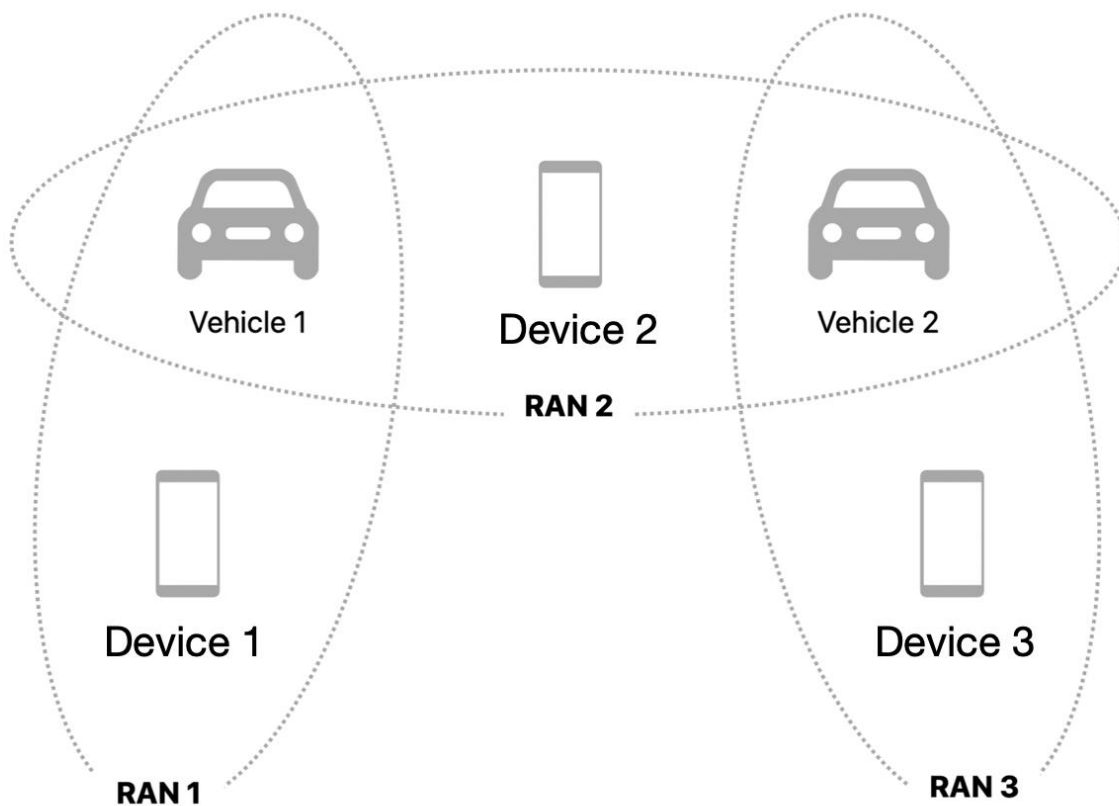
1. A responder can be a logical responder as well as a physical one.
2. A physical responder shall have one UWB module and one or more physical antennas.
3. A logical responder corresponds to one responder role which, for example, could be assigned to a specific UWB module and one specific physical antenna. Hence, a physical responder may include one or more logical responders.
4. The responder-device shall coordinate which logical responders transmit and in which order such that there is no interference between transmissions from two logical responders belonging to the same responder-device.

20.1.1.3 DK Ranging Area Network (RAN)

1. An initiator and a responder-device engaged in a continuous ranging procedure that is characterized by a specific set of parameters is called a ranging session.
2. An initiator and its (one or more) ranging sessions are called a “ranging area network (RAN).” Each RAN is characterized by a timing reference that is established by the initiator. All responder-devices in the same RAN approximate the initiator timeline (there is no assumption of global synchronization here, see Section 20.3).

3. A single RAN shall have one initiator only and may have more than one responder-device (e.g. a device ranging with two vehicles). In the example shown in Figure 20-1, this is represented by Device 2, Vehicle 1, and Vehicle 2 which all belong to RAN 2.
4. Each responder-device may have a different number of logical responders (e.g. one vehicle may have 7 responders and another vehicle in the same RAN may have 5 responders).
5. A single responder-device may belong to multiple RANs, e.g. a single vehicle ranging with two different devices. In the example shown in Figure 20-1, this is represented by Vehicle 2 which belongs to RAN 2 controlled by Device 2 and RAN 3 controlled by Device 3.

Figure 20-1: Digital Key RAN Concept.



20.1.1.4 Inter-/Intra- RAN Interference and Resource Management

Each responder on a responder-device can reliably predict its allowed transmission window and there will be no interference between responders on the same responder-device. However, given the above definitions, three possible performance degradation scenarios can be identified:

1. Inter-RAN Interference:
 - This may be caused by actual over-the-air collisions of packets from different RANs.

• This is the normal mode of operation and should be expected as there is no assumption of coordination between different RANs. The impact of these collisions may be mitigated by employing the ranging round hopping strategy defined below (see Section 20.4).

2. Intra-RAN Resource Conflict:

• A resource conflict may occur when the initiator would have to serve two different ranging exchanges (to two different responder-devices) at the same time.

• This scenario may be resolved at the initiator by prioritizing one of the ranging sessions involved. The target ranging round may be used for the ranging session with the highest priority and the initiator shall skip the round for the other ranging sessions. The impact of skipped ranging round appears like a failed ranging round and may be mitigated by employing the ranging round hopping strategy explained in Section 20.4.

3. Inter-RAN Resource Conflict:

• A resource conflict occurs when a responder would have to serve ranging rounds from two (or more) different RANs at the same time.

• This scenario may be resolved by prioritizing one RAN and skipping the round for the other RANs. The prioritization is determined by the involved responder-device.

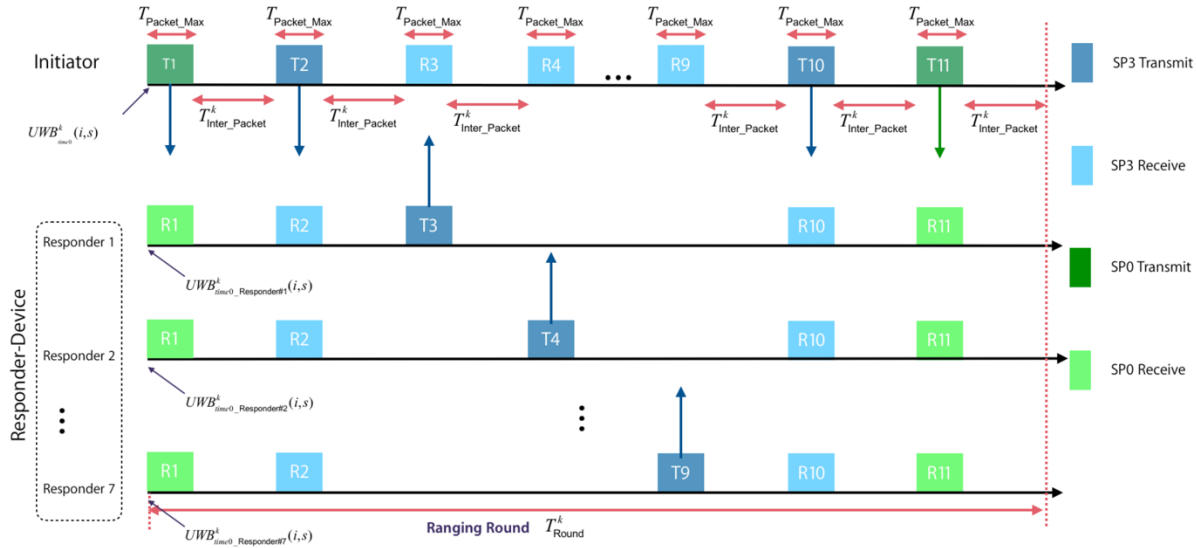
• For the initiators of the skipped RANs, this appears like a failed reception of the responses for the current ranging exchange and the responder-device hops to a different round (see Section 20.4).

The criteria for determining the priority is left to the device and the vehicle manufacturer. In all subsequent sections and text below, unless otherwise stated, a responder is understood to mean a logical responder.

20.2 MAC Time Grid

The DK UWB ranging protocol is a one-to-many (O2M) ranging protocol between the initiator and the responder-device. Figure 20-2 shows an example for the packet exchange in the O2M ranging protocol for an initiator ranging with a responder-device with seven responders. The symbols used in the figure are defined in subsequent text.

Figure 20-2: Digital Key One-to-Many Ranging Protocol.



The DK UWB ranging protocol is a block-based ranging approach as described in [33]. The protocol uses a ranging block structure where each ranging block is T_{Block}^k in time duration (it is explained later in this section how the block timing is established within the RAN). Each ranging block is divided into N_{Round}^k ranging rounds that are long enough to carry out one complete ranging exchange. The length T_{Round}^k of the ranging rounds is session specific.

• First, the following global parameters are defined:

- T_{Chap} : Unit of time for the MAC protocol. All durations (except for $T_{\text{Packet_Max}}$ defined below) of the MAC protocol are integer multiples of T_{Chap} .

$$T_{\text{Chap}} = \frac{1}{3} \text{ms} = 400 \text{ RSTU}$$

where the RSTU is defined in [33] as $416 / (499.2 \text{ MHz}) \gg 833.33 \text{ ns}$.

- $T_{\text{Block_Min}}$: Minimum ranging block duration, which is set to 96ms, is an integer multiple of T_{Chap}

$$T(\text{Block_Min}) = 288 \times T_{\text{Chap}} = 96\text{ms}$$

- One RAN consists of one initiator and $k = 1, \dots, K$ responder-devices.
- Each responder-device is assigned at most one active ranging session in the RAN.
- The k -th ranging session (which is associated with the k -th responder-device) is identified with a unique $UWB_Session_ID^k$ that is assigned during the negotiation phase.

- The k -th ranging session is defined relative to a specific time reference UWB_{time0}^k . This time reference is defined by the initiator with respect to its clock base. This time reference defines the beginning of a series of consecutive ranging blocks. Given that there is no assumption of global synchronization, Section 20.3 describes how that time reference is established at the responders.

- The length of each ranging block for the k -th session is given by:

$$T_{Block}^k = N_{RAN}^k \times T_{Block_Min}$$

where N_{RAN}^k is the RAN Multiplier. It is a session specific parameter that is used to control the maximum ranging frequency of the corresponding session in a given RAN (see Section 20.5.2 and G.1). It is set during the negotiation phase between the initiator and the responder-device. Every ranging session shall have a ranging block duration that is greater than or equal to T_{Block_Min} .

- The $(i+1)$ -th ranging block of the k -th ranging session starts at:

$$UWB_{time0}^k + i \times T_{Block}^k \quad i = 0, 1, 2, \dots$$

- Each block is divided into a number of slots. The length of each slot T_{Slot}^k shall be greater than or equal to the sum of the maximum UWB packet transmission time and the maximum processing time required by the UWB receiver to process such a packet (these are indicated by T_{Packet_MAX} and $T_{Inter_Packet}^k$ in Figure 20-2)⁹. The duration of the slot in terms of T_{Chap} is expressed as:

$$T_{Slot}^k = N_{Chap_per_Slot}^k \times T_{Chap}$$

where $N_{Chap_per_Slot}^k$ is determined during the negotiation phase (see Section 20.5.2).

- A sequence of $N_{Slot_per_Round}^k$ consecutive slots constitutes a ranging round of length T_{Round}^k .

The number of slots per ranging round shall be greater than or equal to the number of packet exchanges required for one complete ranging exchange between the initiator and the responder-device.

$$T_{Round}^k = N_{Slot_per_round}^k \times T_{Slot}^k = N_{Slot_per_round}^k \times N_{Chap_per_Slot}^k \times T_{Chap}$$

⁹ Given that the initiator and the responder-device may be using two different UWB radios with different processing capabilities, the two radios may support different values of slot durations. The duration of the appropriate slot is selected from the set of common values during the negotiation phase between the initiator and the responder-device according to the protocol described in Section 20.5.2.

- Each block of the k -th ranging session is divided into N_{Round}^k consecutive ranging rounds, where

$$N_{\text{Round}}^k = \frac{T_{\text{Block}}^k}{T_{\text{Round}}^k}$$

- The $(s+1)$ -th ranging round in the $(i+1)$ -th ranging block of the k -th ranging session shall start at

$$UWB_{\text{time}0}^k + i \times T_{\text{Block}}^k + s \times T_{\text{Round}}^k \quad i = 0,1,2,3 \dots \quad s = 0,1,2, \dots, (N_{\text{Round}}^k - 1)$$

- Each of the ranging sessions ($k=1,2, \dots, K$) is assigned a pseudo-random hopping sequence

$$H^k = \{S_0^k, S_1^k, \dots, S_i^k\} \quad \text{with } 0 \leq S_i^k \leq (N_{\text{Round}}^k - 1)$$

where S_i^k denotes the index of the ranging round in ranging block i that starts at

$$UWB_{\text{time}0}^k + i \times T_{\text{Block}}^k + S_i^k \times T_{\text{Round}}^k \quad i = 0,1,2,3 \dots \text{ and } 0 \leq S_i^k \leq (N_{\text{Round}}^k - 1)$$

S_i^k also referred to as $\text{Round_Idx}^k(i)$ in later sections of this document to denote the ranging round index of $(i+1)$ -th ranging block selected pursuant to the hopping configuration. The selection of the hopping configuration, i.e. adaptive or continuous, and hopping sequence will be made according to the selection made during the ranging session setup described below. Additionally, Appendix G.2 shows the computation for

the hopping sequence H^k . The hopping sequence shall be used, along with the hopping mode and the hopping flag to compute the ranging round index in the current ranging block. See Section 20.4 below for details on the hopping flag and ranging round index determination.

For details on the ranging packet exchange initiation process and the hopping configuration selection, see Section 20.5.2.

- Given the above definitions, the average ranging frequency of the k -th ranging session is

$$f_{\text{Ranging}}^k = \frac{1}{T_{\text{Block}}^k} \text{ } \pounds \frac{1}{96\text{ms}} @ 0.4166667 \text{ Hz}$$

The achievable ranging block durations T_{Block}^k for a given N_{RAN}^k is $N_{\text{RAN}}^k \times 96\text{ms}$.

- For the $(i+1)$ -th ranging measurement in the k -th ranging session, the ranging round with index $\text{Round_Idx}^k(i)$ is divided into $N_{\text{Slot_per_Round}}^k$ consecutive slots of equal length

$$T_{\text{Slot}}^k = N_{\text{Chap_per_Slot}}^k \times T_{\text{Chap}}$$

The slot with index m shall start at time

$$UWB_{time}^k + i \times T_{Block}^k + Round_{Idx}^k(i) \times T_{Round}^k + m \times T_{Slot}^k$$

$$i = 0,1,2,3, \dots \quad 0 \leq Round_{Idx}^k(i) \leq (N_{Round}^k - 1) \quad m = 0,1,2, \dots, (N_{Slot_per_Round}^k - 1)$$

- The first $N_{Packets}^k$ slots (out of $N_{Slot_per_Round}^k$) of the ranging round with index $Round_Idx^k(i)$ (slots $0,1,2,\dots,N_{Packets}^k - 1$) shall be mapped by the initiator and responder-device to transmit and receive UWB ranging packets (see the example in Table 20-2). The remaining $N_{Slot_per_Round}^k - N_{Packets}^k$ slots shall be left unmapped.
- The initiator and the responder-device shall send the ranging packets as close as technically possible to the beginning of the mapped slots.
- Neither the initiator nor the responder-device shall start transmission before the beginning of a slot.
- If a resource conflict between two (or more) ranging sessions in the same RAN occurs, the initiator shall resolve the conflict by prioritization.

The MAC grid described above is illustrated in Figure 20-3. In this figure and throughout this section, the following notation is used:

$UWB_{time0}^k(i,s,m)$ is the UWB time reference for the $(m+1)$ -th slot of the $(s+1)$ -th ranging round of the $(i+1)$ -th ranging block in the k -th ranging session.

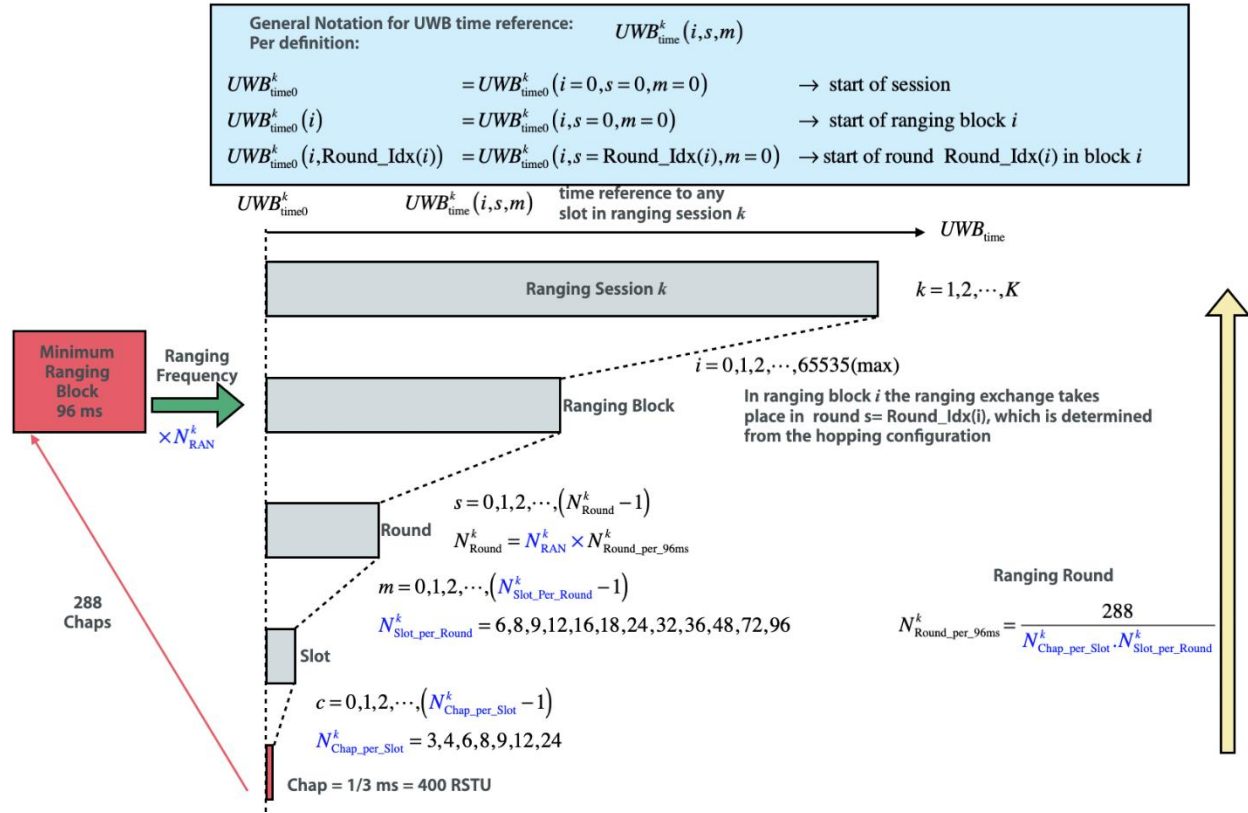
$$UWB_{time0}^k = UWB_{time0}^k(i=0, s=0, m=0) \rightarrow \text{start of session}$$

$$UWB_{time0}^k(i) = UWB_{time0}^k(i, s=0, m=0) \rightarrow \text{start of ranging block } i$$

$$UWB_{time0}^k(i,s) = UWB_{time0}^k(i,s, m=0) \rightarrow \text{start of ranging round } s \text{ in block } i$$

1

Figure 20-3: Overview of MAC Grid (See Table G- 1).



2

3

Table 20-1: Example MAC Configuration.

Parameter	Vehicle 1 (k=1)	Vehicle 2 (k=2)
N_{RAN}^k	3	2
$N_{Chap_per_Slot}^k$	6	3
$N_{Responder}^k$	2	3
T_{Block}^k	288 ms	192 ms
T_{Round}^k	12 ms	8 ms
N_{Round}^k	24	24
$f_{Ranging}^k$	3.47 Hz	5.21 Hz

4

20.3 MAC Time Grid Synchronization

As stated above, each ranging session is defined relative to a specific time reference UWB_{time0}^k , which is defined relative to the initiator's internal clock. The time reference UWB_{time0}^k establishes the MAC time grid for the k -th ranging session.

Given that there is no assumption of tight clock-level synchronization between the initiator and the responder-device, the MAC time grid shall be estimated, with reasonable accuracy, at each responder (at the responder-device). This allows each responder (at the responder-device) to determine when to expect to receive the Pre-Poll message from the initiator, when to send its response message back to the initiator, and when to expect to receive the Final and Final_Data messages from the initiator.

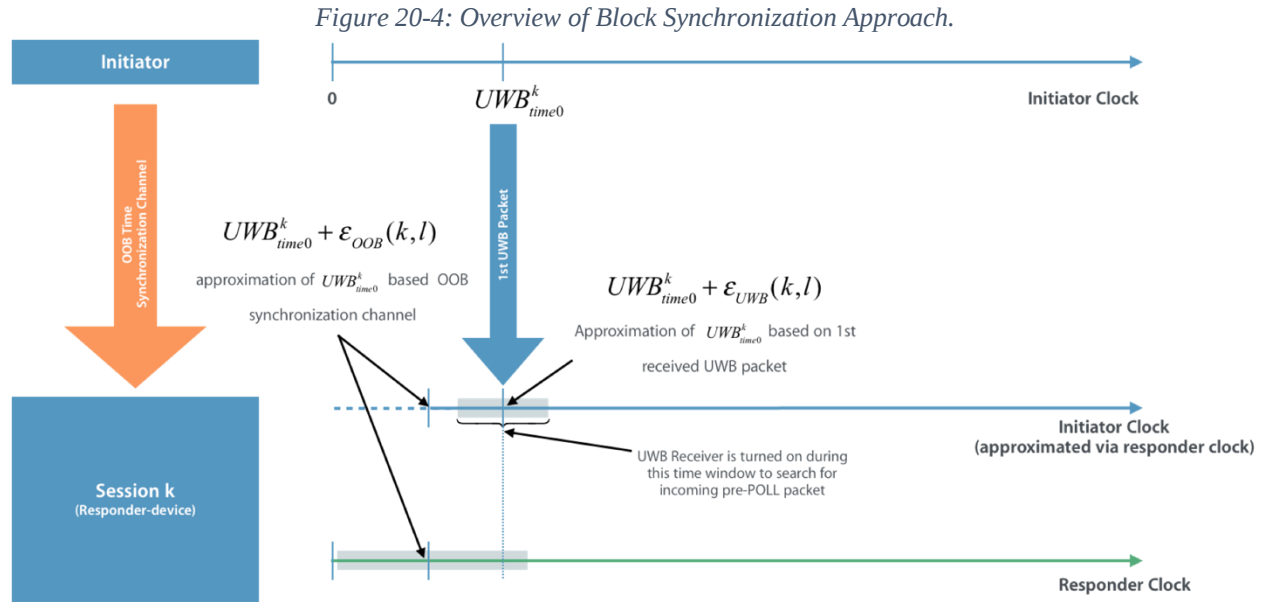
The approximate MAC time grid is established at each responder as follows:

- During the ranging session negotiation, the initiator shall send the value of UWB_{time0}^k to the responder-device that is to be associated with the k -th ranging session.
- Each responder may approximate the MAC time grid of the initiator by:
 - The responder uses a dedicated out-of-band (OOB) clock synchronization (e.g. via the Bluetooth LE control channel). By using an OOB clock synchronization method, the initiator can inform the responder-device when to expect the ranging session to start, i.e. the approximate MAC grid time reference $UWB_{time0_Responder\#l}^k$ for the session for the $(l+1)$ -th responder, $l=0,1,2, \dots, N_{Responder}^k - 1$. The responders then turn on their UWB receivers and start looking for the first UWB packet.
 - Upon receiving UWB packets, the responders further use the incoming UWB packets from the initiator to continuously refine the estimation of the MAC time grid under the assumption that each incoming packet transmission was not started by the initiator before the beginning of its dedicated slot.
 - The time-of-flight (ToF) calculations shall remain unaffected from the MAC grid timings.

For the k -th ranging session, the approximate MAC grid and time reference established at the l -th responder is denoted by $UWB_{time0_Responder\#l}^k$ which is related to the initiator time reference by

$$UWB_{time0_Responder\#l}^k = UWB_{time0}^k + \epsilon(k, l).$$

The error $\epsilon(k, l)$ depends on whether an incoming UWB packet ($\epsilon(k, l) = e_{UWB}(k, l)$) or an OOB clock synchronization protocol ($\epsilon(k, l) = e_{OOB}(k, l)$) is used to approximate the MAC time grid. Characterization of this error and techniques to minimize it are out of scope for this specification. A high-level description of the block timing synchronization process is shown in Figure 20-4.



20.4 Hopping Flag and Round Index Determination

As stated above, the initiator in the k -th ranging session in any given RAN shall start the UWB ranging procedure in the first ranging round (ranging round 0) in the first ranging block (ranging block 0) by default. This assumes that the responder-device has either achieved block synchronization by OOB method or, if not, is permanently listening for Pre-Poll messages.

At the initiator, and assuming no resource conflict occurs, the $Hop_Flag^k(i+1)$ and

$Round_Idx^k(i+1)$ for the next ranging block (ranging block $i+1$) will depend on the hopping mode selected during ranging session setup as follows:

- If the hopping mode is set to “no hopping,” then the initiator shall continue to use the same ranging round in ranging block i and $Round_Idx^k(i+1)$ is set as:

$$Round_Idx^k(i+1) = Round_Idx^k(i) = 0$$

and $Hop_Flag^k(i+1)$ is set to 0. Note that in this case, $Hop_Flag^k(i+1)$ is irrelevant to the receiver and shall be ignored.

- If the hopping mode is set to “continuous hopping,” the initiator uses the S_{i+1}^k -th ranging round

$$Round_Idx^k(i+1) = S_{i+1}^k$$

and $Hop_Flag^k(i+1)$ is set to 1. Again, $Hop_Flag^k(i+1)$ is irrelevant to the receiver and shall be ignored.

- If the hopping mode is set to “adaptive hopping,” then at the initiator, the $Hop_Flag^k(i+1)$ and $Round_Idx^k(i+1)$ for the next ranging block (ranging block $i+1$) shall be set as follows:

- If the initiator determines that the round is clean, i.e. no interference and ranging in the current round is successful, the initiator shall stay in the current round and set as:

$$Hop_Flag^k(i+1)=0 \quad \text{and} \quad Round_Idx^k(i+1)=Round_Idx^k(i)$$

- If the initiator determines that there is some interference on the current round or if the ranging is not successful, the initiator shall hop to a different round:

$$Hop_Flag^k(i+1)=1 \quad \text{and} \quad Round_Idx^k(i+1)=S_{i+1}^k$$

The initiator shall send the $Hop_Flag^k(i+1)$ and $Round_Idx^k(i+1)$ for next ranging block to the responder-device as part of the payload packet carrying time stamps $Final_Data$ at the end of the ranging sequence.

At the responder-device, the $Hop_Flag^k(i+1)$ and $Round_Idx^k(i+1)$ for subsequent ranging rounds shall be set as follows:

if $Final_Data$ packet is received

{

$$Hop_Flag^k(i+1)=Final_Data.Hop_Flag$$

if ($Hop_Flag^k(i+1)=0$)

{

$$\text{set } Round_Idx^k(i+1) = Round_Idx^k(i)$$

}

elseif ($Hop_Flag^k(i+1)=1$)

{

$$\text{set } Round_Idx^k(i+1) = S_{i+1}^k$$

}

}

else

{

$$\text{set } Round_Idx^k(i+1) = S_{i+1}^k.$$

1 }

2

3 The initiator may decide to trigger the hopping (i.e. setting Hop_Flag=1) based on the
4 interference level or packet collision rate that it measures over the current exchange and/or the
5 number of responses that it receives back from the responder-device. In general, the exact criteria
6 for triggering hopping at the initiator is out of scope of this specification and is left to each
7 device manufacturer.

8 *Note:* The only requirement in any such criteria is that if the device does not receive any
9 responses from the responder-device, the device shall trigger hopping.

10 **20.5 MAC Protocol**

11 20.5.1 *Ranging Exchange Sequence*

12 This section details the DK MAC protocol for a double-sided, two-way ranging with a three-
13 packet exchange between the initiator and each responder of the responder-device. Figure 20-5
14 shows the UWB message flow for the MAC protocol between an initiator and $N_{\text{Responder}}^k$

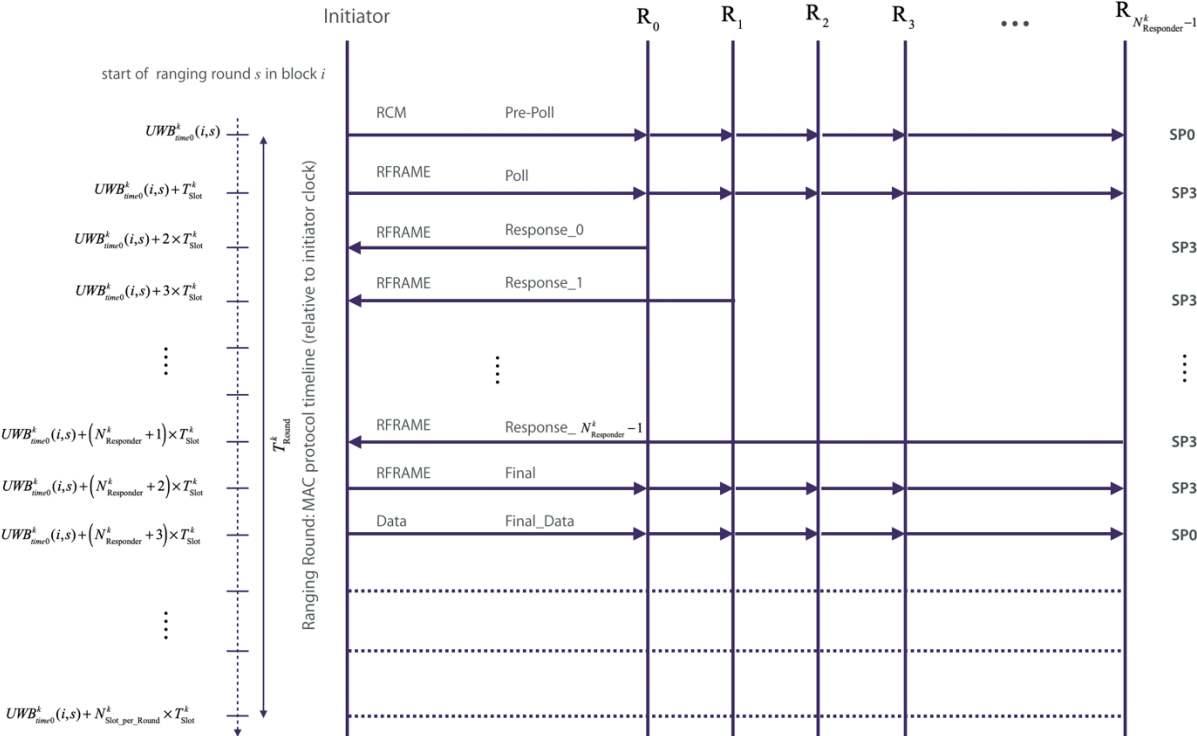
15 responders in the (s+1)-th ranging round in the (i+1)-th ranging block with $N_{\text{Slot_per_Round}}^k$ slots.

16 While the figure shows the MAC protocol according to the MAC timeline relative to the initiator
17 clock and time reference, this protocol shall be carried out according to the respective responder-
18 device's time reference as described in Section [20.3](#). Table 20-2 shows the mapping of the

19 $N_{\text{Responder}}^k$ UWB ranging packets to the $N_{\text{Slot_per_Round}}^k$ slots.

1

Figure 20-5: Example of UWB Message Flow for MAC Protocol.



2

Table 20-2: Mapping of Slots to UWB Ranging Packets.

Slot Index	Mnemonic	STS Packet Format	Sender
0	Pre-POLL	SP0 : Data Packet	Initiator
1	POLL	SP3 : RFRAME	Initiator
1+1	Response_0	SP3 : RFRAME	Responder
1+2	Response_1	SP3 : RFRAME	Responder
...	...	SP3 : RFRAME	Responder
1+ N_{Responder}^k	Response_ N_{Responder}^k - 1	SP3 : RFRAME	Responder
1+ N_{Responder}^k +1	Final	SP3 : RFRAME	Initiator
1+ N_{Responder}^k +2= N_{Packets}^k	Final_Data	SP0 : Data Packet	Initiator
1+ N_{Responder}^k +3	Slot not used	N/A	N/A
...	Slot not used	N/A	N/A
N_{Slot_per_Round}^k -1	Slot not used	N/A	N/A

4 The MAC protocol sequence under the normal mode of operation (i.e. both initiator and
5 responder-device are engaged in an active ranging session) is as follows:

1. The first packet sent by the initiator is a data packet called Pre-POLL message at UWB_{time0}^k . Note that, as mentioned above, the ranging starts in ranging round 0 in ranging block 0 by default when a ranging session is started or recovered. This first packet is a SP0 packet. Its payload shall include:
 - a. $UWB_Session_ID^k$: ID of the current ranging session.
 - b. $STS_Index^k(i, Round_Idx^k(i), POLL)$: STS index of the succeeding POLL message.
 - c. i : ranging session current block index
 - d. $Hop_Flag^k(i)$: Hopping flag as set from the previous ranging exchange. If current ranging round is the first ranging round after start or recovery of the ranging session, then $Hop_Flag^k(i=0)$ is set to '0.' Note that this field is only analyzed by the receiver if the hopping mode is set to "adaptive hopping."
 - e. $Round_Idx^k(i)$: Round index for the (i+1)-th ranging block as set in Final_Data packet of the previous ranging exchange (i-th ranging block). If the current ranging round is the first in the started or recovered ranging session, then $Round_Idx^k(i=0)$ is set to '0.'

The Pre-POLL message and its parameters are defined in Table 20-3 and Table 20-4. All values of the parameters listed in Table 20-4 shall be encoded in little endian format.

Table 20-3: Pre-Poll Request Message and its parameters.

UWB MAC Message	UWB MAC Message ID	Parameters
Pre-POLL	1	UWB_Session_ID, Poll_STS_Index, Ranging_Block, Hop_Flag, Round_Index

Table 20-4: The definition of parameters in the Pre-Poll Message.

Parameters	Length (bytes)	Value	Description
UWB_Session_ID	4	0 - (2 ³² -1)	ID of the UWB ranging session
Poll_STS_Index	4	0 - (2 ³¹ -1)	STS index of the succeeding POLL message
Ranging_Block	2	0 - 65535	Session's Index of current ranging block
Hop_Flag	1	0:No Hopping 1:Hopping	Hop flag for current ranging block as set from the ranging exchange in the previous ranging block For no hopping configuration this field is always 0 For continuous hopping configuration this field is always 1 (except for the first ranging block with index 0 after start or recovery of a ranging session).

Parameters	Length (bytes)	Value	Description
Round_Index	2	0-65535	The ranging round index for the current ranging block as set from the ranging exchange in the previous ranging block. This field is set to 0 in first ranging block where we always start in ranging round 0.

- 1 2. After a period of T_{Slot}^k measured from UWB_{time0}^k , the initiator shall start sending a POLL
- 2 message. This shall be an SP3 type packet (see Section 21).
- 3 3. Each responder l ($l = 0, \dots, N_{\text{Responder}}^k - 1$) shall send its response packet in its dedicated
- 4 response slot.
- 5 a. Each message shall be sent as a SP3 type packet. The index to the STS used is
- 6 related to the POLL message sent in step 2 above (see Section 20.6).
- 7 b. If any of the responders does not receive the POLL message in the current ranging
- 8 round from the initiator, that responder shall not transmit during its dedicated
- 9 response slot.
- 10 4. If at least one valid response has been received by the initiator:
- 11 a. After $(N_{\text{Responder}}^k + 2)$ slots, the initiator shall send its Final message. This packet is
- 12 an SP3 packet and it completes the time-of-flight measurements. The initiator
- 13 may optionally send its Final message together with the following Final_Data
- 14 packet, if no valid response has been received. In the case the Final message and
- 15 Final_Data packet is skipped:
- 16 • The hopping mode for both the initiator and the responder shall be triggered if
- 17 the adaptive hopping mode is active.
- 18 • The Frame Counter value shall not be incremented for the skipped Final Data
- 19 packet.
- 20 b. After $(N_{\text{Responder}}^k + 3)$ slots, the initiator shall complete the ranging exchange by
- 21 sending a Final_Data packet to the responder-device. This packet is an SP0 type
- 22 packet and the payload shall include the following:
- 23 • $UWB_Session_ID^k$: id of the current ranging session.
- 24 • i : index of the current ranging block.
- 25 • $Hop_Flag^k(i+1)$: Hopping flag to be used in ranging block $i+1$. Note that this field is only
- 26 relevant if the hopping configuration is set to adaptive hopping.
- 27 • $Round_Idx^k(i+1)$: ranging round index of the next ranging exchange in ranging block $i+1$.
- 28 • $STS_Index^k(i, Round_Idx^k(i), FINAL)$: STS index of the preceding FINAL message.
- 29 • $Final_Time_Stamp^k(i)$: the time stamp for the Final message. This time stamp shall be
- 30 calculated as the difference between the RMARKER of the initiator POLL message and the
- 31 RMARKER of the initiator Final message

- $N_{\text{Rx_Responses}}$: Number of received responses in the current ranging exchange.
- $[Time_Stamps^k(l)]$: All the ranging measurement time stamps for all responders, up to $N_{\text{Responder}}^k$, whose valid ranging responses have been received at the initiator. The initiator may additionally add the time stamp data of responders where no valid response has been received. The size of the payload depends on the number of added time stamp data. The time stamp data for each responder shall include the following fields:
 - Responder_Index l , $l = 0, 1, \dots, N_{\text{Responder}}^k - 1$, corresponding to responder slot l (slot index $2+l$ in the ranging round).
 - Ranging_Timestamp_Responder value for $(l+1)$ -th responder. The time stamp value shall be calculated as the difference between the RMARKER of the initiator POLL message and the RMARKER of the $(l+1)$ -th responder's response message (see Section 21.6 for details). In the case of no valid response was received from the responder but the time stamp data are added for this responder, the time stamp value shall be set to 0.
 - Ranging_Timestamp_Uncertainty_Responder parameter.
 - Ranging_Status_Responder parameter is set to 0x0 (success) in case of a valid response was received, otherwise this parameter is set to one of the values shown in Table 20-7 to show the reason for the error.

The Final_Data message and its parameters are defined in Table 20-5 and Table 20-6.

Table 20-5: Final_Data Message and its parameters.

UWB MAC Message	UWB MAC Message ID	Parameters
Final_Data	2	UWB_Session_ID, Ranging_Block, Hop_Flag, Round_Index, Final_STS_Index, Ranging_Timestamp_FINAL_TX, Number_Ranging_Responders, Responder_Index, Ranging_Timestamp_Responder_ l , Ranging_Timestamp_Uncertainty_Responder_ l , Ranging_Status_Responder_ l , ...

Table 20-6: The definition of parameters in the Final_Data Message.

Parameters	Length (bytes)	Value	Description
UWB_Session_ID	4	0–($2^{32}-1$)	ID of the UWB ranging session
Ranging_Block	2	0–65535	Index of current ranging Block
Hop_Flag	1	0: No Hopping 1: Hopping	Hop Flag for next ranging Block For no hopping configuration this field is always 0

Parameters	Length (bytes)	Value	Description
			For continuous hopping configuration this field is always 1
Round_Index	2	0 – 65535	The ranging round in which the ranging cycle will be executed in the next ranging block
Final_STS_Index	4	0 – (2 ³¹ -1)	STS index of the preceding Final message.
Ranging_Timestamp_FINAL_TX	4	Range = 0 to 0xFFFFFFFF Timestamp = N × 1/128 × 1/499.2e6 sec Time Range = 0 to 67.21 ms	Time difference between POLL and Final messages transmit times at the initiator.
Number_Ranging_Responders	1	0–255	Number of timestamps to follow in this message
Responder_Index	1	0–255	Index of Responder <i>l</i> whose timestamp data is referred to in the Ranging_Timestamp_Responder_ <i>l</i> , Ranging_Timestamp_Uncertainty_Responder_ <i>l</i> , and Ranging_Status_Responder_ <i>l</i> fields
Ranging_Timestamp_Responder_ <i>l</i>	4	Range = 0 to 0xFFFFFFFF Timestamp=N × 1/128 × 1/499.2e6 sec Time Range = 0 to 67.21 ms	Time difference between POLL and RESPONSE of Responder <i>l</i> , as received by the initiator.
Ranging_Timestamp_Uncertainty_Responder_ <i>l</i>	1	See Section 6.9.1.7 in [33]	Range of values from 1.5 cm–3.6 m at various confidences
Ranging_Status_Responder_ <i>l</i>	1	See Table 20-7	The status for the response frame from the responder. See the corresponding description for the ranging status in Table 20-7
...	...	...	Repeat parameters Responder_Index to Ranging_Status_Responder_ <i>l</i> for each responder whose time stamp data has been added (up to Number_Ranging_Responders times)

- 1 All values of the parameters listed in Table 20-6 shall be encoded in little endian format.
- 2 Additionally, with the above definition of the Final_Data message and with a 127 maximum
- 3 payload size, the maximum number of responders $N_{\text{Responder}}^k$ that can be involved in a single
- 4 ranging round is 10 responders. In case $N_{\text{Chap_per_Slot}}^k = 24$, then the maximum number of
- 5 responders $N_{\text{Chap_per_Slot}}^k$ in a single ranging round is limited to 7 due to the maximum time stamp
- 6 value for the Final message. A vehicle may have any number of responders N_v (physical and/or
- 7 logical) and this number may be >10 responders. However, the vehicle shall set the value of the
- 8 number of responders it wants to involve in each ranging round $N_{\text{Responder}}^k$ to a number that is less
- 9 than or equal to 10 in the ranging session setup procedure. It is up to the vehicle to determine
- 10 which set of its responders it will involve in the ranging exchange in each ranging round.
- 11 The Ranging_Status_Responder_*l* parameter uses the values shown in Table 20-7.

Table 20-7: Ranging Status of a Responder.

Ranging Status	Value	Description
Success	0x0	Successful receipt and transmission of a packet
Transaction overflow	0x1	RESPONSE SP3 frame from this responder cannot be processed by the device
Transaction expired	0x2	No RESPONSE SP3 frame was received from this responder
Incorrect frame	0x03	The RESPONSE SP3 frame received from this responder was not correct
Reserved	0x04-0xFF	

20.5.2 Ranging Session Setup

The following procedure describes how the MAC parameter negotiation shall be carried out between the initiator and the responder-device. First, the responder-device and the initiator shall perform a ranging capability exchange step over Bluetooth LE link, where the responder-device sends a RC-RQ message to the initiator and the initiator replies with a RC-RS message, as defined in Sections 19.3.1.1 and 19.3.1.2, respectively. Once this exchange is successful, the following negotiation steps shall be carried out over the Bluetooth LE control channel to setup the ranging session:

- Vehicle initiates the ranging session setup by sending the Ranging Session Request message defined in Section 19.3.1.3 with the following content:
 - Selected_Protocol_Version^k*: selected protocol version from the ranging capability exchange.
 - Selected_UWB_Config_ID^k*: selected UWB configuration from the ranging capability exchange.
 - UWB_Session_ID^k*: id of the current ranging session.
 - Selected_PulseShape_Combo*: device selected single pulse shape combination from list of common supported ones.
 - Channel_BitMask*: $\{CH_IDX\}_{Responder}$ list of available UWB RF channels.
- Device responds by sending the Ranging Session Response message (see Section 19.3.1.4) with the following contents:
 - RAN_Multiplier*: RAN multiplier N_{RAN}^k sets the minimum ranging block duration for the k -th ranging session that can be supported by the initiator (i.e. sets the maximum ranging frequency that can be supported by the initiator for this ranging session)

$$T_{Block_RAN}^k = N_{RAN}^k \cdot T_{Block_Min} = N_{RAN}^k \cdot 96ms$$

$$f_{Ranging_RAN}^k = \frac{1}{T_{Block_RAN}^k} = \frac{10.4166667}{N_{RAN}^k} Hz$$

- 1 • *Slot_BitMask*: $\left\{ N_{\text{Chap_per_Slot}} \right\}_{\text{Initiator}}$ list of slot durations supported by the initiator for
2 the session expressed as specified in Section 19.3.1.4 and Table G-1. Note that
3 initiator may prefer to use slot durations that are not necessarily the minimum (by
4 excluding the minimum $N_{\text{Chap_per_Slot}}$ from the list) to accommodate other constraints
5 it may have, such as co-existence.
- 6 • *SYNC_Code_Index_BitMask*: $\left\{ \text{SYX_IDX} \right\}_{\text{Initiator}}$ list of available UWB preamble
7 sequences that the responder-device may choose from.
- 8 • *Selected_UWB_Channel CH_IDX*: selected UWB RF channel.
- 9 • *Hopping_Config_Mask*: $\left\{ H_Config_Seq \right\}_{\text{Initiator}}$ bitmask indicating the hopping
10 configuration (no hopping, continuous, adaptive) and hopping sequences supported
11 by the device.
- 12 3. Responder-device calculates/selects:
 - 13 • *Session_RAN_Multiplier* $N_{\text{RAN_S}}^k$: the responder-device selects an N_{RAN}^k value that is
14 greater than or equal to the value sent by the initiator to achieve a desired ranging
15 interval of T_{Block}^k that is an integer multiple of $T_{\text{Block_Min}}^k$:
16
$$N_{\text{RAN_S}}^k = \frac{T_{\text{Block}}^k}{T_{\text{Block_Min}}^k} \left(N_{\text{RAN_S}}^k \geq N_{\text{RAN}}^k \right) \quad \text{and} \quad f_{\text{Ranging}}^k = \frac{1}{T_{\text{Block}}^k} = \frac{10.4166667}{N_{\text{RAN_S}}^k} \text{ Hz}$$
 - 17 • *Number_Chaps_per_Slot* $N_{\text{Chap_per_Slot}}^k$: shortest slot duration that is common between
18 slot durations supported by initiator $\left\{ N_{\text{Chap_per_Slot}} \right\}_{\text{Initiator}}$ and slot durations supported
19 by the responder-device $\left\{ N_{\text{Chap_per_Slot}} \right\}_{\text{Responder}}$:
20
$$N_{\text{Chap_per_Slot}}^k = \min \left(\left\{ N_{\text{Chap_per_Slot}} \right\}_{\text{Initiator}}, \left\{ N_{\text{Chap_per_Slot}} \right\}_{\text{Responder}} \right)$$
 - 21 • *Number_Slots_per_Round* $N_{\text{Slot_per_Round}}^k$: responder-device selects the number of slots
22 that is greater than or equal to $\left(N_{\text{Responder}}^k + 4 \right)$ out of all possible values of slots
23 corresponding to $N_{\text{Chap_per_Slot}}^k$ (see Table G-1 for details).
 - 24 • *SYNC_Code_Index* $\left\{ \text{SYX_IDX} \right\}_{\text{Responder}}$: list of UWB preamble sequences that the
25 responder-device selected to use. This set is a subset of the list sent by the initiator.
 - 26 • *Hopping_Config_Mask*: $\left\{ H_Config_Seq \right\}_{\text{Responder}}$ bitmask indicating the hopping
27 configuration and hopping sequences selected by responder-device.
- 28 4. Responder-device uses the “Ranging Session Setup Request (RSS-RQ)” message as
29 defined in Section 19.3.1.5 to send the following to the initiator:

- 1 a. $N_{\text{RAN_S}}^k, N_{\text{Chap_per_Slot}}^k, N_{\text{Responder}}^k, N_{\text{Slot_per_Round}}^k, \{\text{SYN_IDX}\}_{\text{Responder}},$
2 $\{H_Config_Seq\}_{\text{Responder}}$

3 5. Initiator selects the number of rounds per block for the ranging session N_{Round}^k as

$$4 \qquad N_{\text{Round}}^k = \frac{288 \times N_{\text{RAN_S}}^k}{N_{\text{Chap_per_Slot}}^k \times N_{\text{Slot_per_Round}}^k}$$

5
6 Appendix G.1 shows a list of all valid/possible numbers of rounds for different combinations of
7 $(N_{\text{Chap_per_Slot}}^k, N_{\text{Slot_per_Round}}^k)$ for each 96 ms of block duration.

8 6. Initiator selects the preamble SYNC code index SYNC_Code_Index^k . See Section
9 [21.4.1](#) for the definition of BPRF SYNC.

10 7. Initiator uses the “Ranging Session Setup - Response (RSS-RS)” message as defined in
11 Section 19.3.1.6 to send the following back to the responder-device:

- 12 a. $\text{STS_Index0}^k, \text{UWB}_{\text{time0}}^k, \text{HOP_Key_RW}^k$
- 13 b. SYNC_Code_Index^k

14 Figure 20-6 shows an example of the negotiation handshake used to set the MAC parameters.
15 The following applies to MAC parameters negotiation:

- 16 1. $N_{\text{Chap_per_Slot}}^k = 8$ is mandatory and shall be supported by all devices.
17 • This avoids a deadlock situation due to a possibly disjoint configuration parameters
18 capabilities between the initiator and the responder-device during negotiation.
- 19 2. Responder-device will accept any N_{RAN}^k chosen by the initiator, even if it does not allow
20 it to realize the desired ranging frequency.
21 • The initiator shall set N_{RAN}^k as small as possible to meet the quality of service (QoS)
22 constraints to meet the functional requirement.

Figure 20-6: Example of MAC Parameter Negotiation and Setting.

															Tchap	0.333 msec		
Step 1: RS-RQ ("Ranging Session Request") - Vehicle initiates ranging session setup																		
Selected_Protocol_Version																		
Selected_UWB_Config_Id																		
UWB_Session_Id																		
Step 2: RS-RS ("Ranging Session Response") - Smartphone sets supported minimum block duration and Chaps per Slot																		
N_RAN	2	input value												T_Block_RAN	192.0 msec			
N_Chap_per_Slot	3	4	6	8	9	12	24										f_Ranging_RAN	5.208 Hz
{N_Chap_per_Slot}_SP	FALSE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	select supported values										
SYNC_Code_Index_BitMask																		

20.5.3 MAC Control Channel (Over Bluetooth LE)

A MAC control channel shall be implemented with following requirements:

- Control channel shall be available for negotiation or re-negotiation of UWB parameters (e.g. ranging session setup as in Section 20.5.2).
- The control channel shall be session specific and shall be available before the start of, during the entire lifetime of, and after the termination of the ranging session.
- A RAN will have as many UWB control channels as it has responder-devices.
- The control channel shall be carried over the Bluetooth LE connection.
- Control traffic with respect to the k -th ranging session shall be referenced with the $UWB_Session_ID^k$.
- Timings during an active ranging session shall be referenced via the appropriate MAC time grid indices (e.g. block index, round index, slot index, or mapped STS indices) of this ranging session.
- The control channel shall be triggered by a channel ID in the Bluetooth LE L2CAP channel. Initiator or responder-device may trigger the control channel and can request a change of UWB parameters.

20.5.4 UWB MAC Configuration

All SP0 type packets sent according to the above MAC protocol shall include a MAC header (MHR), MAC payload and MAC Footer (MFR) fields. This section describes how the contents of the SP0 packets are constructed.

Figure 20-7: SP0 Packet Content.

MHR (23 Octets)	MAC Payload Frame Payload (variable) + MIC (8 octets)	MFR (2 Octets)
-----------------	--	----------------

20.5.4.1 MHR Field

The contents of the MHR field according to Section 7.2 of [31] are defined in Table 20-8. All values of the parameters listed in Table 20-8 shall be encoded in little endian format.

Table 20-8: The Contents of the MHR Field.

MAC parameter	Length (bytes)	Description
Frame Control	2	Describes how the MAC frame is configured
Sequence Number	0	Sequence number is not used. Sequence Number Suppression bit field in the Frame Control shall be set to 1
Destination PAN ID	0	Destination PAN ID not present
Destination Address	2	Destination Short Address
Source PAN ID	0	Source PAN ID not present
Source Address	0	Source Address not present
Auxiliary Security Header	10	Describes security configuration for data payload encryption and message identification authentication (MIC)
Vendor Specific Header IE	7	Used to signal the UWB message ID and the UWB message length
Header Termination IE	2	Header Termination IE of Type HT2. See 7.4.1 of [31]

The Frame Control field depends on the specific MAC header configuration. See [31] for more information about the individual parameters and their definitions. The Frame Control field is defined in Table 20-9.

Table 20-9: The Contents of the Frame Control Field.

Frame Control	Length (bits)	Value	Description
Frame Type	3	0b001 (data)	Data frame being transmitted
Security Enabled	1	0b1	Every packet has security enabled
Frame Pending	1	0b0	Frame Pending Flag is always set to 0.
AR	1	0b0	This flag indicates whether this frame requires acknowledgment (0b1) or not (0b0). Always set to 0 as neither Pre-Poll or Final Data packets require an acknowledgment back from the vehicle
PAN ID Compression	1	0b1	Source Addressing Mode set to 0, Destination Addressing Mode set to 2 (short address), and Destination PAN ID not present this corresponds to the 4th entry of Table 7-2 in [31].

Frame Control	Length (bits)	Value	Description
Reserved	1	0b0	
Sequence Number Suppression	1	0b1	Sequence number is disabled
IE Present	1	0b1	Header IE Used.
Destination Addressing Mode	2	0b10	Destination Short Address
Frame Version	2	0b10	802.15.4 frames for data and acknowledgement.
Source Addressing Mode	2	0b0	Source PAN ID and Source Address field are not present

The Auxiliary Security Header defines the security configuration. The Auxiliary Security Header is defined in Table 20-10. Note that the Auxiliary header is not relevant to SP3 packets (Poll and Final messages) since SP3 packets do not get security processing due to the fact they do not carry MAC data frames (payload, MHR, and MFR). Therefore, The Auxiliary Security header is applicable to SP0 packets only (Pre-Poll and Final_Data). All values of the parameters listed in Table 20-10 shall be encoded in little endian format.

Table 20-10: The Contents of the Auxiliary Security Header.

Auxiliary Security Header	Length (bytes)	Value	Description
Security Control Field	See below		
Security Level	3 bits	0b110 (ENC-MIC-64)	Security level for authentication of MAC header and data payload
Key Identifier Mode	2 bits	0b10, Key is determined explicitly from the 4-byte Key Source field and Key Identifier field	Key identification mode
Frame Counter Suppression	1 bit	0b0, enable frame counter	Frame counter is suppressed or not
ASN in Nonce	1 bit	0b0, ASN in not in nonce	Is ASN in nonce or not
Reserved	1 bit	0b0	
Frame Counter	4 bytes	0x00000000-0xFFFFFFFF	Counter that increments for each transmitted packet from a source
Key Identifier Field	See below		
Key Source	4 bytes	0x00000000-0xFFFFFFFF	Key Source derived from UAD
Key Index	1 byte	0xAA	

The content of the Header IE defined the Vendor Specific Header IE which shall be used to signal the UWB message ID and the UWB message length carried by the corresponding UWB packet.

The Vendor Specific Header IE is defined in Table 20-11. All parameter values listed in Table 20-11 shall be encoded in little endian format.

1

Table 20-11: The Contents of the Vendor Specific Header IE.

Vendor Specific Header IE	Length (bytes)	Value	Description
Length	7 bits	5 (bit 0 - bit 6)	Length of the IE content in bytes
IE ID	8 bits	0 (bit 7 - bit 14)	Vendor Specific Header IE
Type	1 bit	0 (bit 15)	Header IE
Vendor OUI	3 bytes	0x04DF69	OUI of Car Connectivity Consortium per IEEE registration authority (See http://standards-oui.ieee.org/oui.txt).
Vendor Specific Information	2 Bytes	Byte 1: 0x00 - 0xFF Byte 2: 0x00 - 0xFF	Byte 1: UWB MAC message ID of message sent in this UWB packet 0x01: Pre-Poll UWB MAC message ID 0x02: Final_Data UWB MAC message ID 0x03-0xFF: Reserved for future use Byte 2: UWB MAC message payload length in bytes sent in this UWB packet excluding the MHR, MFR, and MIC fields.

20.5.4.2 MHR Source and Destination Address

The Source Addressing Mode shall be set to 0 in the Frame Control Field of the MHR. This indicates that the Source Address and the Source PAN ID shall not be present in the MHR. The Destination Addressing Mode shall be set to 2 and the PAN ID Compression shall all be set to 1 in the Frame Control field of the MHR. This indicates that the Destination Short address shall be present, and the Destination PAN ID shall not be present in the MHR. The Destination Short Address along with Source Long Address and Key Source fields shall be generated according to address rotation procedure -as described in Section 22.

20.5.4.3 MAC Payload

The MAC payload field is of variable length and includes the MAC frame payload plus the 64-bit message integrity check (MIC) field

20.5.4.4 MFR Field

MAC footer shall be according to Section 7.2.10 of [31], which states that the MAC footer shall contain a Frame Check Sequence (FCS) field. For the BPRF mode in the HRP UWB PHY, the FCS contains a 16-bit ITU-T CRC. The FCS is calculated over the MHR and MAC payload part of the frame; these parts together are referred to as the calculation fields. This 2-byte FCS is calculated using the following standard generator polynomial of degree 16:

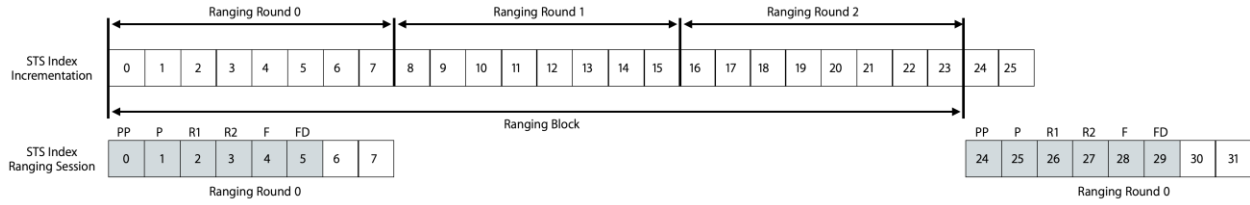
$$G_{16}(x) = x^{16} + x^{12} + x^5 + 1$$

20.6 STS Index Incrementation

The STS index parameter (which needs to change for every slot in ranging block regardless of whether that slot is used or not) plays an important role for the secure ranging operation. This section defines the STS index incrementation and the mapping of the STS index onto the packets

of a ranging session within the MAC layer framework. The basic principle of the STS incrementation is shown in Figure 20-8.

Figure 20-8: Basic Principles of STS Incrementation.



(The example uses $N_{\text{Responder}}^k = 2$, $N_{\text{Slot_per_Round}}^k = 8$, $N_{\text{Round}}^k = 3$. Note that $N_{\text{Round}}^k = 3$ is not a supported number of rounds in the current design but is used in this example for illustration purpose only.)

20.6.1 Rules for STS Index Incrementation

The following rules shall be used for incrementing the STS index:

1. The STS index is managed separately for each ranging session of a given RAN.

2. A k -th ranging session starts at time UWB_{time0}^k .
3. The STS index of the k -th ranging session is referenced to the start of this ranging session:

STS_Index0^k is mapped to ranging round 0 of ranging block 0. That is, for $i=0$ and

$$Round_Idx^k(i=0)=0$$

$$STS_Index^k(i=0, Round_Idx^k(i=0)=0) = STS_Index0^k$$

4. Within a ranging round, the STS index increments in every slot even if that slot is unmapped (i.e. not used).
5. The incrementation scheme is applied for every ranging round (even if it is not used). This results in:

- a. The STS index is incremented by $N_{\text{Slot_per_Round}}^k$ in every ranging round.
- b. The STS index is incremented by $N_{\text{Slot_per_Round}}^k \cdot N_{\text{Round}}^k$ in every ranging block.

6. The STS index is incremented for all slots in the round. However, data packets (Pre-POLL and Final_Data) reference the STS index in their payload as follows:
 - a. Pre-POLL message shall carry the reference to the STS index that is used in the following POLL message.
 - b. Final_Data message shall refer to the STS index of the preceding FINAL message.

20.6.2 STS Incrementation and Packet Mapping within a Ranging Round

Within any ranging round s in ranging block i , the STS index increments for every slot (relative to the first STS index in the round):

- Pre_POLL $STS_Index^k(i, s)$
- POLL $STS_Index^k(i, s) + 1$
- Response_0 $STS_Index^k(i, s) + 2$
- Response_1 $STS_Index^k(i, s) + 3$
- ...
- Response_ $N_{Responder}^k - 1$ $STS_Index^k(i, s) + N_{Responder}^k + 1$
- Final $STS_Index^k(i, s) + N_{Responder}^k + 2$
- Final_Data $STS_Index^k(i, s) + N_{Responder}^k + 3$

20.6.3 Calculation of STS Index

The following formulas are used to calculate the STS indices for ranging round $Round_Idx^k(i)$ in ranging block i :

$$\begin{aligned}
 STS_Index^k(i, Round_Idx^k(i), Pre_Poll) &= STS_Index0^k + (i \times N_{Round}^k + Round_Idx^k(i)) \times N_{Slot_per_Round}^k \\
 STS_Index^k(i, Round_Idx^k(i), POLL) &= STS_Index0^k + (i \times N_{Round}^k + Round_Idx^k(i)) \times N_{Slot_per_Round}^k + 1 \\
 STS_Index^k(i, Round_Idx^k(i), R_0) &= STS_Index0^k + (i \times N_{Round}^k + Round_Idx^k(i)) \times N_{Slot_per_Round}^k + 2 \\
 STS_Index^k(i, Round_Idx^k(i), R_1) &= STS_Index0^k + (i \times N_{Round}^k + Round_Idx^k(i)) \times N_{Slot_per_Round}^k + 3 \\
 &\vdots \\
 STS_Index^k(i, Round_Idx^k(i), R_{N_{Responder}^k - 1}) &= STS_Index0^k + (i \times N_{Round}^k + Round_Idx^k(i)) \times N_{Slot_per_Round}^k + N_{Responder}^k + 1 \\
 STS_Index^k(i, Round_Idx^k(i), Final) &= STS_Index0^k + (i \times N_{Round}^k + Round_Idx^k(i)) \times N_{Slot_per_Round}^k + N_{Responder}^k + 2 \\
 STS_Index^k(i, Round_Idx^k(i), Final_Data) &= STS_Index0^k + (i \times N_{Round}^k + Round_Idx^k(i)) \times N_{Slot_per_Round}^k + N_{Responder}^k + 3
 \end{aligned}$$

21 UWB PHY [WCC3]

21.1 Introduction

The UWB PHY uses a waveform based on an impulse radio signal using band-limited pulses. UWB PHY is primarily used for ranging but may also be used for data communication. UWB PHY described in this specification is based on HRP UWB PHY in [33].

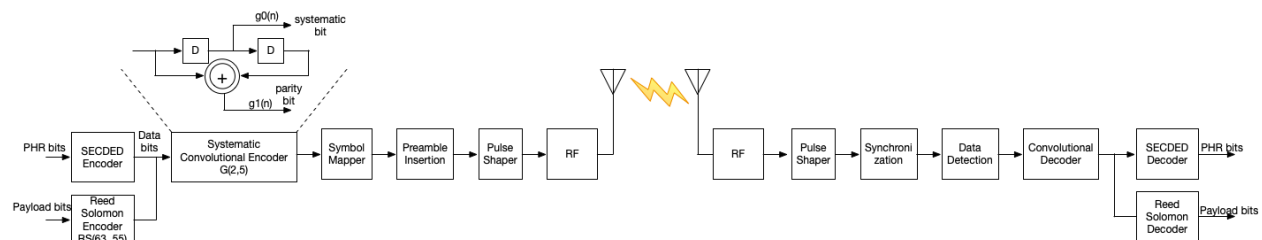
This section describes interoperable enhanced ranging devices (ERDEVs) featuring enhanced immunity to attack. The main enhancement for secure time of flight is the inclusion of a Scrambled Timestamp Sequence (STS) in the basic PPDU format. Note that the STS is included in the basic PPDU format in the BPRF mode as specified in [33].

The IEEE standard defines a very flexible UWB PHY. The flexibility in IEEE standard is offered by adaptation of parameters like preamble for synchronization (SYNC) lengths, preamble codes, Start of Frame Delimiter (SFD) lengths/codes, pulse repetition frequencies (PRFs), and data rates. This specification, however, does not require to implement all the parameters and modes that are specified in [33]. Please refer to appropriate sections on mandatory and optional PHY modes for secure ranging, ERDEVs on both sides of the link shall pre-negotiate the specific parameters that will be used for a secure ranging session. The negotiations of secure ranging parameters can be done at higher layers or using Bluetooth LE.

21.2 UWB PHY Block Diagram

Figure 21-1 shows the block diagram for IEEE 802.15.4z HRP UWB PHY.

Figure 21-1: Block diagram for IEEE 802.15.4z HRP UWB PHY.



21.3 UWB Packet Format

Figure 21-2 and Figure 21-3 illustrate the two, packet types for UWB configurations: STS packet type 3 (SP3) and STS packet type 0 (SP0).

For SP3, the Deterministic Random Bit Generator (DRBG) STS engine shall generate 4096 bits to create the STS waveform and there is no PHR or data field. Note that packet configurations to be used for UWB ranging (and also for UWB data communication) shall be known a-priori. The packet configurations to be used between two nodes shall be exchanged over Bluetooth LE link. However, the length of the data field in SP0 is flexible.

Figure 21-2: SP3

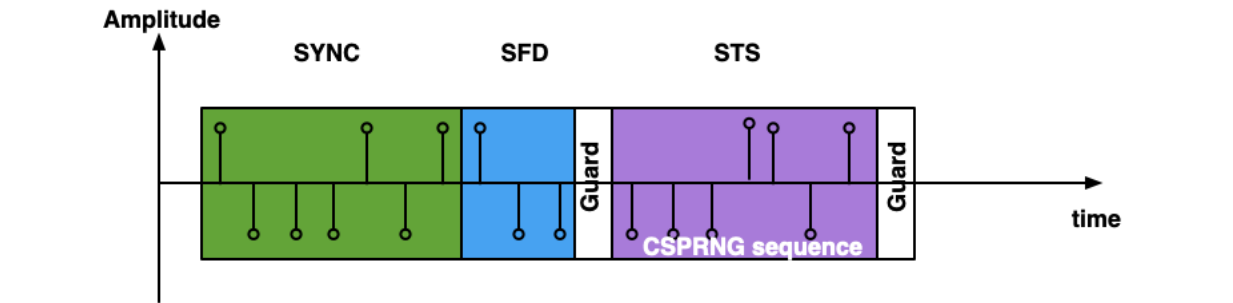
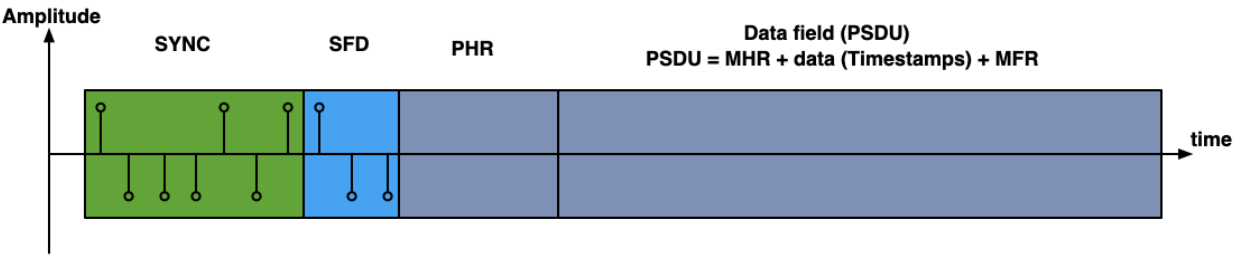


Figure 21-3: SP0



SP3 is intended for use cases where the participants in the secure ranging exchange are known to each other such that information about the source and/or the destination are implicit in the knowledge of what STS is used for transmission and reception between the connected devices, respectively.

A nominal mean PRF of at least 62.4 MHz shall be used for all packets during a ranging session. Nominal mean PRFs lower than 62.4 MHz are not required for ERDEVs.

21.4 UWB Frame Elements

Only IEEE 802.15.4z HRP BPRF UWB SP0 and SP3 packets are used in this specification. SP0 packet consists of preamble (SYNC), SFD, PHY header (PHR), and payload (PSDU or Data field). Supported configurations are referenced as UWB_Config as shown in Table 21-1. Each of the fields is described in detail in the subsections below.

Table 21-1: Supported UWB Configurations.

DK protocol			M/O	STS Packet Type 3 (SP3) IEEE 802.15.4z parameters				STS Packet Type 0 (SP0) IEEE 802.15.4z parameters						
UWB configuration	UWB Channel (IEEE 802.15.4)	SYNC preamble code (IEEE 802.15.4)		corresponding IEEE BPRF Mandatory Set	phyHrpUwbPsr	SFD (phyHrpUwbSfd-Selector)	STS (segment × symbols)	corresponding IEEE 802.15.4z BPRF Set	phyHrpUwbPsr	SFD (phyHrpUwbSfd-Selector)	STS	phyHrpUwbPhr-DataRate	PHR bit rate	PSDU bit rate
0	5, 9	9,10,11,12 Optional: 13-16, 21-24	M	N/A	64	ternary (0)	1× 64	BPRF #1	64	ternary (0)	no	DRBM- _LP	0.85 Mbps	6.81 Mbps
1	5, 9	9,10,11,12 Optional: 13-16, 21-24	Device: M Vehicle: O	BPRF #4	64	binary (2)	1× 64	BPRF #2	64	binary (2)	no	DRBM- _LP	0.85 Mbps	6.81 Mbps

21.4.1 BPRF SYNC

SYNC is used to aid receiver algorithms related to AGC settling, timing acquisition, coarse and fine frequency recovery, packet and frame synchronization. [31] defines four mandatory preamble lengths for SYNC: a default preamble, a short preamble, a medium preamble, and a long preamble. The length of default preamble is 64 symbols, short preamble is 16 symbols, medium preamble is 1024 symbols, and long preamble is 4096 symbols. The PHY as specified in this specification shall support 64 symbols short preamble.

ERDEV in this specification, shall support ternary code with length 127, even though, ternary code with length 127 is optional in [31]. Each preamble code is a sequence of code symbols drawn from a ternary alphabet $\{-1,0,1\}$ and is selected for use because of their perfect periodic autocorrelation properties.

In this specification, the ERDEV shall operate on channels 5 and 9, and hence the support for the four length-127 ternary code sequences shown in Table 21-2 are mandatory, though they are optional in [31].

Table 21-2: The Mandatory Length-127 Ternary Code Sequences.

Code index	Ternary Code Sequence	UWB channels
9	+00+000-0--00--+0+0+00-++0+0000++-000+00-00--0+0+0--0+++0++000+- 0+00-0++-0+++00-+0+0+0-0+++-+000000+00000-+0000-0-000--+	5,9
10	++00+0-+00+00+000000-000-00--000-0-+-0-0-+00000+-00++0-0+00--00++- +0+-0+0000-0-0-0-++-+0+0+0+000-+0+++000-----+0000+++0--	5,9
11	--+0000+00--00000-0+0+0+-0+00+00+0-00-+++00+000-+0+0-0000++++-+0+-- 0+-0+--0-000+0-+00+0+----000-000000-+00+-0++000+-00++-0-0	5,9
12	--0+++000000-0+0-0-+0---+0+00-+0+0+0+0+000-00-00-+00+-+000+-0-++0- 0+++0-00-0++0+0+0++-00+000+-000-0--0000-0000--0+00000+--	5,9

The Code index 9, 10, 11, 12 shall be supported.

Support for code indexes 13-16 and 21-24 ternary code sequences as specified in [31] is optional in this specification.

21.4.2 BPRF SFD

SFD is added to the UWB frame to establish frame timing at the receiver. Only length 8 SFD is used in this specification:

The short ternary SFD shall be $[0 +1 0 -1 +1 0 0 -1]$ spread by the preamble symbol S_i , where the leftmost bit shall be transmitted first in time. This is as specified in [31]. To improve performance, the binary sequence: $[-1 -1 -1 +1 -1 -1 +1 -1]$ should be supported per [33].

21.4.3 Scrambled Timestamp Sequence (STS)

UWB PHY supports Secure Ranging mechanisms to prevent different forms of attacks carried out by a “rogue” device injecting signal energies with the objective of making ranging receivers misread the distance between associated ranging peers. Scrambled Timestamp Sequence (STS) provides an AES-generated ranging waveform to protect against range measurement spoofing. DRBG based STS is resilient to various types of distance manipulation attacks like Brute force attack, Cicada attack, preamble injection attack and preamble EDLC. Details on how STS sequence is generated is specified in [33]. The number of bits used for STS is specified in Table 21-1. STS bits are mapped to pulses as follows:

1. Bit 1 is mapped to a negative pulse
2. Bit 0 is mapped to a positive pulse

In the BPRF (64 MHz PRF) mode, there is one pulse every 8 chips (spreading = 8)

21.4.4 BPRF PHR

A PHR, as shown in Figure 21-4, shall be added after the SHR preamble. The PHR consists of 19 bits and conveys information necessary for a successful decoding of the packet to the receiver. The PHR contains information about the data rate used to transmit the PSDU, the duration of the current frame’s preamble, and the length of the frame payload. Additionally, six parity check bits are used to further protect the PHR against propagation channel errors.

Figure 21-4: PHR bit assignment.

Bits: 0–1	2–8	9	10	11–12	13–18
Data Rate	Frame Length	Ranging	Reserved	Preamble Duration	SECDED

It is mandatory for RDEVs and ERDEVs to support transmission and reception of PRF64 PHRs at 850 Kbit/s data rate. The BPRF PHR shall use Single Error Correct Dual Error Detect (SECDED) encoding followed by systematic convolutional encoder with generator polynomial $G(2,5)$ as shown in Figure 21-1. The SECDED field is a set of six parity check bits that is used to protect the PHR from errors caused by noise and channel impairments. The SECDED bits are a simple Hamming block code that enables the correction of a single error and the detection of two

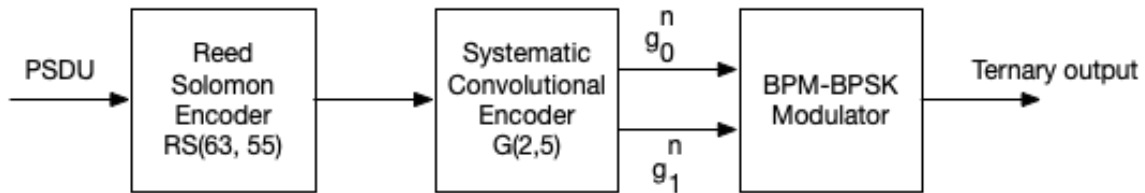
errors at the receiver. The SECDED bit values depend on PHR bits 0–12 and are computed as follows:

$b_{18} = \text{XOR}(b_1, b_0, b_8, b_6, b_4, b_3, b_{10}, b_{11})$
 $b_{17} = \text{XOR}(b_0, b_6, b_5, b_3, b_2, b_9, b_{10}, b_{12})$
 $b_{16} = \text{XOR}(b_1, b_8, b_7, b_3, b_2, b_9, b_{10})$
 $b_{15} = \text{XOR}(b_8, b_7, b_6, b_5, b_4, b_9, b_{10})$
 $b_{14} = \text{XOR}(b_{12}, b_{11})$
 $b_{13} = \text{XOR}(b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8, b_9, b_{10}, b_{11}, b_{12}, b_{14}, b_{15}, b_{16}, b_{17}, b_{18})$

21.4.5 Standard Compliant Data Field

The Data field is transmitted using the 6.8 Mbps data rate at a mean PRF of 62.4 MHz and encoded as shown in Figure 21-5.

Figure 21-5: Data Field Encoding for Standard PHY.



The data field shall be formed as follows:

- Encode the PSDU using systematic Reed-Solomon block code, which adds 48 parity bits
- Encode the output of the Reed-Solomon block code using a systematic convolutional encoder. If the Viterbi rate for the modulation is defined as one, then the convolutional encoder is bypassed
- Spread and modulate the encoded block using BPM-BPSK modulation

21.5 Pulse Shape Combinations

A compliant transmitter shall choose between a traditional symmetrical Root-Raised-Cosine Pulse (see Section 21.5.1) and a Precursor-Free Pulse as defined by the requirements in Section 15.4.4 and 15.4.5 of [33](see Section 21.5.2).

21.5.1 Symmetrical Root-Raised-Cosine Pulse – PulseShape 0x0

For the symmetrical Root-Raised-Cosine Pulse, referred to as PulseShape 0x0, the RF envelope (or equivalent baseband pulse amplitude) is expressed by [31]:

$$r(t) = \frac{4\beta}{\pi\sqrt{T_p}} \frac{\cos((1+\beta)\pi t/T_p) + \frac{\sin((1-\beta)\pi t/T_p)}{4\beta(t/T_p)}}{1 - (4\beta t/T_p)^2}$$

where:

β = roll-off factor with a value of 0.45 ± 0.05

T_p = Pulse duration, which is inverse of the chip frequency (1/499.2MHz)

Finite impulse response (FIR) samples of PulseShape 0x0 using β of 0.45 with 32x oversampling per chip and total span of 8 chips are included in [A.1.81.a.i.I.1](#) as informative reference.

21.5.2 Precursor-Free Pulse – PulseShape 0x1

In general, “precursor-free” refers to the pulse’s property to monotonically raise its amplitude to the first peak that also represents its main lobe. PulseShape 0x1 represents a group of pulse shapes that meet the time-domain mask of Figure-89 in [\[33\]](#).

21.5.2.1 Precursor-Free Pulse – PulseShape 0x2

PulseShape 0x2 is a special case within PulseShape 0x1, by applying minimum-phase conversion of PulseShape 0x0 using method described in [\[31\]](#). A Python implementation of the linear-to-minimum phase conversion is included in [I.1](#), and FIR samples for a span of 8 chips at oversampling of 32x and β of 0.45 for PulseShape 0x2 are again included in [I.2](#), both provided as informative references. It is recommended to use PulseShape 0x2 whenever possible if transmitter chooses to implement precursor-free pulse shape.

Table 21-3 summarizes the different pulse shapes that allowed in the specification. However, it is recommended to use PulseShape 0x2.

Table 21-3: Summary of pulse shapes.

PulseShape	Description	bitfield
Symmetrical Root-Raised-Cosine Pulse	Symmetrical Root-Raised-Cosine Pulse, referred to as PulseShape 0x0. The RF envelope or equivalent baseband pulse amplitude is expressed by [31] .	0x0
Precursor-Free Pulse	Group of pulse shapes that meet the time-domain mask of Fig. 89 in [33] .	0x1
	A special case within PulseShape 0x1, by applying minimum-phase conversion of PulseShape 0x0 using method described in [31] .	0x2

21.5.3 PulseShape Combinations

Compliant receivers shall be able to receive and suitably process both symmetrical and precursor-free pulses. It is up to the implementer whether the same receiver configuration is used for PulseShape 0x2 and PulseShape 0x1, or whether reception of PulseShape 0x2 is explicitly optimized via a separate receiver configuration.

Possible transmitter PulseShape combinations (PulseShape_Combo) for the initiator and responder are shown in Table 21-4. The responder’s and initiator’s receive configurations shall be set according to the PulseShape selections made by the initiator’s and responder’s transmitters, respectively.

Table 21-4: Overview of PulseShape_Combo values and the associated transmit pulse shapes.

PulseShape_Combo	Transmit PulseShape	
	Initiator	Responder
0x00	0x0	0x0
0x01	0x0	0x1
0x02	0x0	0x2

PulseShape_Combo	Transmit PulseShape	
	Initiator	Responder
0x10	0x1	0x0
0x11	0x1	0x1
0x12	0x1	0x2
0x20	0x2	0x0
0x21	0x2	0x1
0x22	0x2	0x2

21.6 Ranging Marker

The RMARKER as defined in [33] shall be supported for taking timestamp measurements.

21.7 UWB RF

21.7.1 Operating frequency bands and Channel assignments

The UWB technology as specified in [31] supports three independent bands of operation and a wide frequency range from hundreds of MHz to several GHz:

- The sub-gigahertz band, which consists of a single channel and occupies the spectrum from 249.6 MHz to 749.6 MHz.
- The low band, which consists of four channels and occupies the spectrum from 3.1 GHz to 4.8 GHz.
- The high band, which consists of 11 channels and occupies the spectrum from 6.0 GHz to 10.6 GHz.

In this specification, the ERDEV shall operate in the high band using channel 5 and channel 9. Channel 5 uses center frequency of 6489.6 MHz and channel 9 uses center frequency of 7987.2 MHz. ERDEV that only operates in specific geographic regions where regulatory restrictions prevent the use of specific band(s) may be exempted from this requirement.

21.7.2 Frequency Source Requirements

RF center frequency and the symbol clock frequency shall all be derived from the same reference clock (clk_ref), which is typically implemented as a crystal oscillator (XO). Clock and sampling rates for all transmitter and receiver activities in the PHY and the RF Carrier Frequency shall be tied to the same common clk_ref.

RF Tolerance

Both devices and vehicles shall have a transmit carrier frequency tolerance and pulse timing of less than +/- 20 ppm, as defined in Section 15.4.6 of [36]. The first μ s of the SHR shall be omitted from these measurements to account for transmitter settling time.

1 **clk_ref Tolerance**

2 For the device, clk_ref tolerance should be regulated within ± 10 ppm over operation conditions.

3 For the vehicle, clk_ref tolerance should be regulated within ± 15 ppm over operation conditions.

4

22 UWB SECURITY [WCC3]

This section addresses the security requirements inherent (applicable and enforced) to the UWB-related functionality of the Digital Key feature. The device OEM shall use NFC as a fallback mechanism in case correct operation of the UWB and the BLE module is disturbed; Information about the fallback mechanism should be provided by the device OEM to the user.

22.1 Cryptography

This section details the various cryptographic functions identified in the various security functions covered earlier in the document.

Build STS based on cryptographic DRBG

Section 20 describes:

- MAC protocol (Ranging session and MAC control Channel)
- STS Index Incrementation (rules, incrementation and update)

22.1.1 Deriving the STS Sequence from the DRBG

The derivation of the STS follows the sequence described in Section 15.2.9.1 and Figure 15-7b of [33]. For the purpose of STS generation and in the context of this specification, the following parameters as defined in [33] are mapped to the corresponding parameters as defined in this specification as follows:

$\text{phyHrpUwbStsVCounter} = \text{SaltedHash}[0:32]$

$\text{phyHrpUwbStsVUpper96} = \text{SaltedHash}([64:64]) \parallel ((\text{SaltedHash}[32:32] + \text{STS_Index}) \& 0\text{xffffffff})$

$\text{phyHrpUwbStsKey} = \text{dURSK}[0:128]$

The 32-bit counter is initialized with $\text{phyHrpUwbStsVCounter}$ for each STS generation, that means for each slot. The 32-bit counter is incremented by one for each 128-bit chunk of the STS, that means 31 increments for a 4096-bit long STS.

$\text{SaltedHash}[0:128]$, $\text{STSIndex}[0:32]$, and $\text{Counter}[0:32]$ (the same one as the IEEE 32-bit counter, initialized at 0) are used to derive $\text{phyHrpUwbStsVUpper96}$ and $\text{phyHrpUwbStsVCounter}$.

22.1.2 SP0 Frames Encryption

SP0 shall be encrypted using the AES-CCM* algorithm as described in Section 9.3 of [31]. Unlike the NIST specification for CCM which requires encrypted data length > 0, the IEEE 802.15 CCM* implementation allows authentication-only CCM (as used in [31] with Security Level set to 1, 2, or 3).

STS_index encryption (SP0 Pre-POLL)

- *Algorithm*: AES-CCM* as described in Section 9.3 of [31]
- *Mode*: Encrypted and authenticated [31](Security Level 6 - ENC-MIC64)

- $K(\text{Key}) = \text{mUPSK1}[0:128]$
 - $N(\text{Nonce}) = \text{Source Long Address} \mid \text{Frame Counter} \mid \text{Nonce Security Level}$ as described in Section 9.3.2.1 of [31]
 - Incrementing Frame Counter, STS_Index in payload

Note: The key mUPSK1 is static for the ranging session.

Timestamps encryption (SP0 Final_Data timestamp frame)

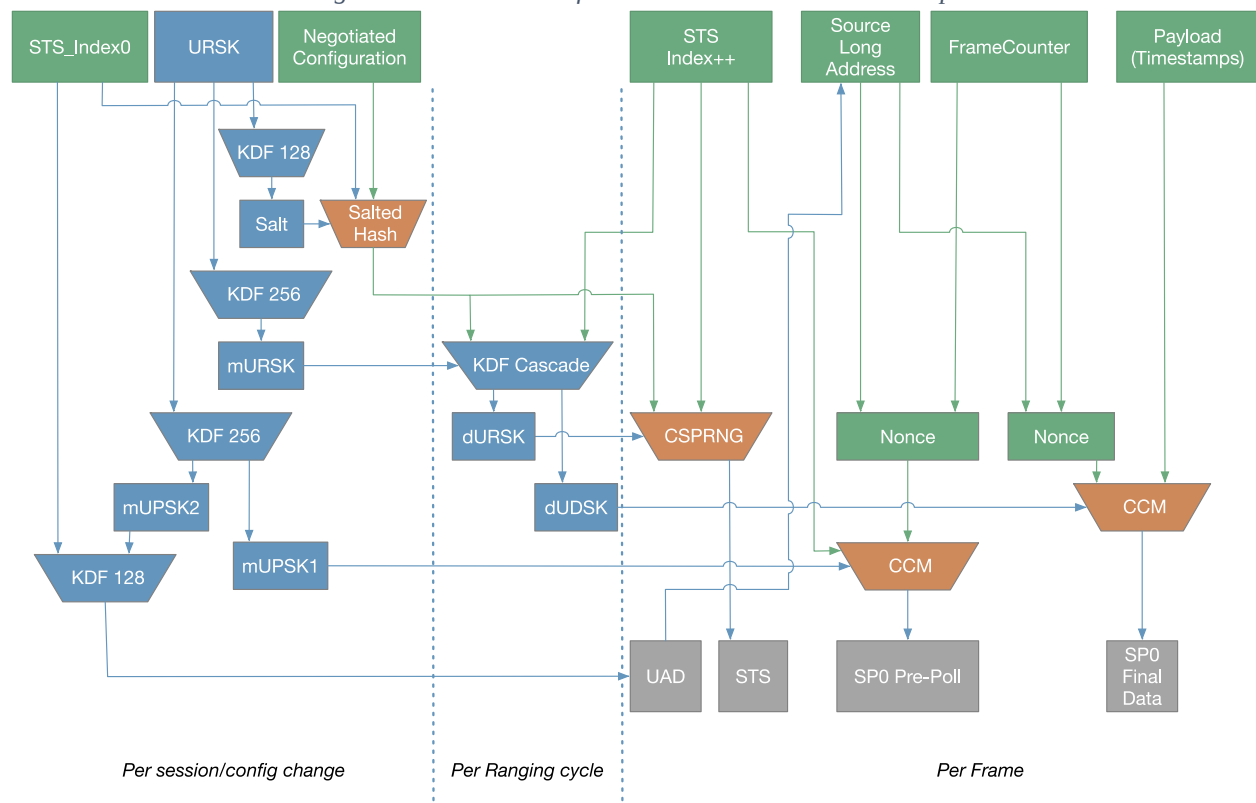
- Algorithm: AES-CCM* per as described in Section 9.3 of [31]
- Mode: Encrypted and authenticated [31](Security Level 6 - ENC-MIC64)
- $K(\text{Key}) = \text{dUDSK}$, where dUDSK is defined later in this subsection
- $N(\text{Nonce}) = \text{Source Address} \mid \text{Frame Counter} \mid \text{Nonce Security Level}$ as described in Section 9.3.2.1 of [31]
- Incrementing Frame Counter

Note: The probability of IV/Key repetition is insignificant since the key dUDSK shall be updated for every ranging block.

22.1.3 Key Derivation Functions (KDFs)

The cryptographic foundations of the protocol shall be established with the following cascaded KDFs as shown in Figure 22-1. Information about specific key derivations is in the subsequent sections.

Figure 22-1: Overview of the KDF used and its relationship.



22.1.3.1 mUPSKs Key Derivation

Purpose: Protection of STS_Index

Occurrence: Once per session

Cryptography:

$mUPSK1[0:128] = CMAC(URSK, 0x00000001 \parallel 0x5550534B \parallel 0x00 \parallel 0x000000 \parallel 0x00000180)$

$mUPSK2 = mUPSK2[128:128] \parallel mUPSK2[0:128]$, where

$mUPSK2[128:128] = CMAC(URSK, 0x00000002 \parallel 0x5550534B \parallel 0x00 \parallel 0x000000 \parallel 0x00000180)$

$mUPSK2[0:128] = CMAC(URSK, 0x00000003 \parallel 0x5550534B \parallel 0x00 \parallel 0x000000 \parallel 0x00000180)$

CMAC = AES256-CMAC as PRF, $r = 32$, $h=128$

Reference: See Section 5.1 of [4] KDF in Counter Mode

Note: ASCII (0x5550534B) = “UPSK”

mUPSK1 is used in order to encrypt the SP0 Pre-POLL.

mUPSK2 is used to derive the KeySource and UWB addresses.

22.1.3.2 SaltedHash Key Derivation

Purpose: Preserve integrity of ranging configuration

Occurrence: Once per ranging configuration negotiation

Cryptography:

$Salt[0:128] = CMAC(URSK, 0x00000001 \parallel 0x53414C54 \parallel 0x00 \parallel 0x000000 \parallel 0x00000080)$

CMAC = AES256-CMAC as PRF, $r = 32$, $h=128$

Reference: See Section 5.1 of [4] KDF in Counter Mode

$PaddedSalt[0:256] = 0x000..0000 \parallel Salt$

$RangingConfiguration = Selected_Protocol_Version \parallel Selected_UWB_Config_Id \parallel UWB_Session_Id \parallel STS_Index0 \parallel Number_Responder_Nodes \parallel Session_RAN_Multiplier \parallel Number_Slot_per_Round \parallel Number_Chaps_per_Slot \parallel Selected_PulseShape_Combo$

$SaltedHash = CMAC(PaddedSalt, RangingConfiguration)$

CMAC = AES256-CMAC as PRF, $r = 32$, $h=128$

Note: ASCII (0x53414C54) = “SALT”

RangingConfiguration parameters are fully described in Section 20.5.2.

22.1.3.3 *mURSK Key Derivation*

Purpose: Ranging Keys Seed derivation

Occurrence: Once per session

Cryptography:

$mURSK = mURSK[128:128] \parallel mURSK[0:128]$, withwhere
 $mURSK[128:128]$
 $= CMAC(URSK, 0x00000001 \parallel 0x5552534B \parallel 0x00 \parallel 0x000000 \parallel 0x00000100)$
 $mURSK[0:128]$
 $= CMAC(URSK, 0x00000002 \parallel 0x5552534B \parallel 0x00 \parallel 0x000000 \parallel 0x00000100)$

$CMAC = AES256-CMAC$ as PRF, $r = 32$, $h=128$

Reference: Section 5.1 of [4] KDF in Counter Mode

Note: ASCII (0x5552534B) = “URSK”

22.1.3.4 *dURSK/dUDSK Keys Derivation (KDF_cascade definition)*

Purpose: Ranging Keys derivation

Occurrence: KDF executed once per ranging cycle (before “POLL”)

Cryptography:

$URSK_KT = URSK_KT[128:128] \parallel URSK_KT[0:128]$, where
 $URSK_KT[128:128] = CMAC(mURSK, 0x00000001 \parallel 0x5552534B5F4B54 \parallel$
 $0x00 \parallel$
 $7'b0 \parallel STS_Index[31] \parallel .. 7'b0 \parallel STS_Index[16] \parallel 7'b0 \parallel STS_Index[15] \parallel ..$
 $7'b0 \parallel STS_Index[1] \parallel 7'b0 \parallel STS_Index[0] \parallel 0x00000100)$
where $7'b0$ represents seven binary zeros and $STS_Index =$
 $STS_Index^k(I, Round_Idx^k(i), Pre_POLL)$

$URSK_KT[0:128] = CMAC(mURSK, 0x00000002 \parallel 0x5552534B5F4B54 \parallel$
 $0x00 \parallel$
 $7'b0 \parallel STS_Index[31] \parallel .. 7'b0 \parallel STS_Index[16] \parallel 7'b0 \parallel STS_Index[15] \parallel ..$
 $7'b0 \parallel STS_Index[1] \parallel 7'b0 \parallel STS_Index[0] \parallel 0x00000100)$

$dURSK[0:128]$
 $= CMAC(URSK_KT, 0x00000001 \parallel 0x5552534B \parallel 0x00 \parallel SaltedHash \parallel$
 $0x000000 \parallel 0x00000080)$

$dUDSK[0:128]$
 $= CMAC(URSK_KT, 0x00000001 \parallel 0x5544534B \parallel 0x00 \parallel SaltedHash \parallel$
 $0x000000 \parallel 0x00000080)$

$CMAC = AES256-CMAC$ as PRF, $r = 32$, $h=128$

Reference: Section 5.1 of [4] KDF in Counter Mode

Note: ASCII (0x5552534B5F4B54) = “URSK_KT”

ASCII (0x5552534B) = “URSK”

ASCII (0x5544534B) = “UDSK”

22.1.4 UWB Tracking Prevention Between Ranging Recoveries

In order to protect against tracking of the same UWB session at session setup or upon recovery (i.e., upon receiving the RR-RS message), the following actions shall be performed:,

- The following values shall be derived:
 - KeySource (4 bytes)
 - Destination Short Address (2 bytes)
 - Source Long Address (8 bytes)
- The Frame Counter shall be reset to 0.

22.1.4.1 UWB Addresses KDF

Purpose: Deriving UWB Addresses (UAD)

Occurrence: At ranging setup or at ranging recovery

Cryptography:

UAD[0:128] = CMAC(mUPSK2, 0x00000001 || 0x554144 || 0x00 ||

STSIndex0[0:32] || 0x00000080)

Some addresses shall be reserved for broadcast use:

- For the KeySourceLow, KeySourceHigh and the destination short address: 0xFFFF and 0xFFFE are reserved values.
- For the source long address: 0xFFFFFFFFFFFFFFFF is a reserved value.

The following method shall be used to flip the upper bit if the calculated UAD matches the reserved value:

```
function SetUpperBitToZeroIfReserved(bytes [X...0]) {  
    if (bytesAreReserved(bytes)) {  
        bytes[X...X-7] = bytes[X...X-7] & 0x7f  
    }  
    return bytes  
}
```

Note: The indexing notation still refers to bit-indexes.

The SetUpperBitToZeroIfReserved remaps the following entries and leaves the others unmodified:

Reserved Address	Remapped Address
0xFFFF	0x7FFFF
0xFFFE	0x7FFE

Reserved Address	Remapped Address
0xFFFFFFFFFFFFFFFF	0x7FFFFFFFFFFFFFFF

KeySourceLow[0:16] = SetUpperBitToZeroIfReserved(UAD[112:16])
KeySourceHigh[0:16] = SetUpperBitToZeroIfReserved(UAD[96:16])
DestinationShortAddress[0:16] = SetUpperBitToZeroIfReserved(UAD[80:16])
SourceLongAddress[0:64] = SetUpperBitToZeroIfReserved(UAD[16:64])
CMAC = AES256-CMAC as PRF, $r = 32$, $h = 128$
Reference: Section 5.1 of [4] KDF in Counter Mode
Note: ASCII (0x554144) = “UAD”
The latest STSIndex0 passed is the one negotiated during session setup or recovery.

22.1.4.2 Key Source for Auxiliary Security Header
The 4-byte Key Source value shall be obtained as follows:
KeySource [0:32] = KeySourceHigh || KeySourceLow

22.2 UWB Ranging

The following applies to the UWB ranging session:

22.2.1 STS Index Management

The STS Index Incrementation shall comply with the following requirements (see Section 20.6).

1. The initial value of STS_Index selected at the start of a given ranging session (referred here as initial STS_Index0). It shall be a random value in the $[0..2^{30}-1]$ range, therefore the STS_Index may be incremented at least $2^{30}-1$ times. The initial STS_Index0 is included in Ranging Session Setup Response (see Section 19.3.1.6).
2. The STS_Index shall be monotonically increasing. Within a given ranging session identified by UWB_Session_ID, the STS_Index shall never go back be decremented or reused between any two different frames.
3. The maximum value of STS_index in a given ranging session shall be $2^{31}-1$. When this maximum value is reached, the ranging session shall be ended and a new session with a new URSK shall be used.
4. The first STS_Index of a recovered ranging session (referred here as recovery STS_Index0) is included in Ranging Recovery Response (See Section 19.3.1.10) or Configurable Ranging Recovery Response (see Section 19.3.1.12). The recovery STS_Index0 shall be greater than the last used STS_Index prior to Ranging Suspension which is within the same ranging session.
5. The STS_Index value included in the Poll_STS_Index parameter of the first Pre-POLL message (received by the vehicle) shall be larger than the initial STS_Index0 or recovery STS_Index0 for the case of a new ranging session or a recovery of a suspended ranging session, respectively.

Re-synchronization

The re-synchronization of the anchors happens on the vehicle side. This mechanism involves the mUPSK1, the STS_index, and a nonce (calculated from data in the MAC header): the device performs the encryption of the frame (SP0) and the vehicle decrypts.

22.3 UWB Module

The pre-derived URSKs are generated by the DK applet. Then, when it becomes active, the URSK or sub keys derived from this URSK (depending on the device architecture) are transferred to the UWB Module.

The UWB Module is notably in charge of the processing of some cryptography operation like PPDU ciphering or derivation of the STS. An implementation shall provide hardware and software protection against logical and certain physical attacks for the full lifecycle of all assets related to the UWB secure ranging.

The transport channel between a SE and the UWB module shall protect confidentiality and integrity. The SE shall provide a secure binding with the UWB module that shall be resistant to manipulation after production. It shall be prevented that any code running on the application processor has access to or can inject data on this channel. The SE shall have the ability to distinguish communication between the UWB chip interface, and other interfaces.

The potential associated with an effective attack on the UWB module and its interface with the SE should be expressed in terms of the total effort required to mount a successful attack as defined by the identified threats. The detailed semantics of such attack potential is to be determined based on definition and related documentation by the DKCTG (e.g. upon conducting a threat analysis exercise).

Appropriate security measures shall be taken to prevent an attacker with expert expertise and bespoke equipment from leveraging a vulnerability to mount a successful exploit on any of the CCC DK Release 3 solution components, that would defeat the security of the use case (passive keyless entry or start engine), given sensitive knowledge of the implemented solution and that a few months time is available for searching vulnerabilities.

23 REFERENCES

The documents listed in this section are included in requirements made in the body of this specification. Knowledge of their contents is required for the understanding and implementation of this specification. If a listing includes a date or a version identifier, only that specific version of the document is required. If the listing includes neither a date nor a version identifier, the latest version of the document is required. External references referred herein should apply to the latest version provided they are backward compatible with the version at the time of writing and are not violating the implementation specified in this specification.

- [1] ISO/IEC 7816-4: Fourth Edition May 2020
- [2] ISO 8601-1:2019: Date and time – Representations for information interchange – Part 1: Basic rules
- [3] RFC 5280 - PKIX Certificate and CRL Profile - May 2008
- [4] NIST SP 800-108 - Recommendation for Key Derivation Using Pseudorandom Functions - May 2005 (Updated August 2022 as NIST SP 800-108r1)
- [5] NIST SP 800-38B - Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication – May 2005 (Updated October 2016)
- [6] BSI TR-03111 - Elliptic Curve Cryptography - Version 2.0
- [7] GPC_SPE_014 - GlobalPlatform - Secure Channel Protocol 03 - Amendment D - Version 1.1.1 or later version
- [8] FIPS PUB 186-4 - Digital Signature Standard - July 2013
- [9] NIST SP 800-38A - Block Cipher Modes of Operation: Methods and Techniques - 2001
- [10] Network Working Group Internet-Draft: SPAKE2+, an Augmented PAKE, draft-bar-cfrg-spake2plus-00, Mar 9, 2020
- [11] Internet Engineering Task Force (IETF): The Scrypt Password-Based Key Derivation Function (RFC 7914), August 2016
- [12] Internet Engineering Task Force (IETF): Randomness Requirements for Security (RFC 4086), June 2005
- [13] RFC 5869 - HMAC-based Extract-and-Expand Key Derivation Function - May 2010
- [14] RFC 5480 - ECC SubjectPublicKeyInfo Format - October 2009
- [15] GlobalPlatform Card Technology – Confidential Card Content Management, Card Specification v2.3 Amendment A Version 1.2 (or higher)
- [16] NFC Analog Technical Specification 2.1 (<https://nfc-forum.org/our-work/specifications-and-application-documents/>) or later version.
- [17] NFC Digital Protocol Technical Specification 2.1 (<https://nfc-forum.org/our-work/specifications-and-application-documents/>) or later version.
- [18] NFC Activity Technical Specification 2.0 (<https://nfc-forum.org/our-work/specifications-and-application-documents/>) or later version.

- 1 [19] NFC Controller Interface Technical Specification 2.1 ([https://nfc-forum.org/our-](https://nfc-forum.org/our-work/specifications-and-application-documents/)
2 [work/specifications-and-application-documents/](https://nfc-forum.org/our-work/specifications-and-application-documents/)) or later version.
- 3 [20] GlobalPlatform Card Technology Card Specification v2.2.1 (or higher)
- 4 [21] GlobalPlatform Card – Contactless Extension Version 1.0 (or higher)
- 5 [22] SEC 1: Elliptical Curve Cryptography Version 2.0, <https://www.secg.org/sec1-v2.pdf>
- 6 [23] ISO/IEC 10118-3 Hashfunctions - Part 3: Dedicated hash-functions
- 7 [24] FIPS PUB 180-4: Specifications for the Secure Hash Standard – 2015
- 8 [25] ETSI TS 102 705 Release 9.2.0 (or higher) Smart Cards; UICC Application
9 Programming Interface for Java Card™ for Contactless Applications,
10 <https://www.etsi.org/standards>
- 11 [26] ETSI TS 102 622 Release 9.4.0 (or higher) Smart Cards; Contactless Front End, HCI
12 (Host Controller Interface), <https://www.etsi.org/standards>
- 13 [27] RFC 3986 – Uniform Resource Identifier (URI): Generic Syntax – January 2005
- 14 [28] GPC_SPE_093 - GlobalPlatform - Secure Channel Protocol '11' - Amendment F -
15 Version 1.1
- 16 [29] Car Connectivity Consortium "Android Digital Key Framework API", Version 1.0, Aug
17 2020
- 18 [30] The Bluetooth Core specification, version 5.0, December 2016
- 19 [31] IEEE Standard for Low-Rate Wireless Networks, 802.15.4-2020, July 2020
- 20 [32] CCC Digital Key Release 2 Specification version 1.1
- 21 [33] IEEE 802.15-4z-2020 - IEEE Standard for Low-Rate Wireless Networks, Amendment 1:
22 Enhanced Ultra Wideband (UWB) Physical Layers (PHYs) and Associated Ranging
23 Techniques, June 2020
- 24 [34] N. Damara-Venkata, B. L. Evans, S. R. McCaslin, “Design of optimal minimal phase
25 filter using discrete Hilbert transform,” IEEE Trans. Signal Proc., vol. 48, pp. 1491-1495,
26 May 2000
- 27 [35] CCC Vehicle OEM Document, D1.1
- 28 [36] Description 15.4z HRP UWB PHY Test Vectors”, IEEE 802.15 document number 15-20-
29 0003-02-004z, 2020.
- 30 [37] RFC 2330 - Framework for IP Performance Metrics,
31 <https://tools.ietf.org/html/rfc2330#section-10>
- 32 [38] Internet Engineering Task Force (IETF): Secure Credential Transfer - draft-secure-
33 credential-transfer-03, <https://datatracker.ietf.org/doc/draft-secure-credential-transfer/>
- 34 [39] NIST Special Publication 800-63B, Digital Identity Guidelines, June 2017 including
35 updates of 03-02-2020
- 36 [40] ITU Telecommunication Standardization Sector - Recommendation ITU-T X.690
37 International Standard 8825-1
- 38 [41] CCC Digital Key Specification R3 v1.1.3
- 39 [42] NIST SP 800-90A – Recommendation for Random Number Generation Using
40 Deterministic Random Bit Generators – June 2015 (NIST SP 800-90Ar1)

- 1 [43] NIST SP 800-90B – Recommendation for the Entropy Sources Used for Random Bit
2 Generation – January 2018 – (NIST SP 800-90B)
- 3 [44] RFC 2560 – X.509 Internet Public Key Infrastructure Online Certificate Status Protocol –
4 OCSP – June 1999
- 5 [45] A comparison of Pedestrian Dead-Reckoning algorithms using a low-cost MEMS IMU –
6 August 2009 – <https://ieeexplore.ieee.org/document/5286542>
7

Appendix A.

A.1. Standard Transaction Cryptography Flow

Listing A-1: Standard Transaction Cryptography Flow

```
Vehicle key pair:
vehicle public key X: F47EB42A771052580C086EFDAAA3084AA3FF7A67CE23393A0373C63487DF1A63
vehicle public key Y: 7D1FB34B2D2E7D5C8F92097A0619B5C5CC6C5850AF74C019EBBEC4273358AA94
vehicle private key: 86B9E3843D949890FD50E49C8542DB575BAC41D344F17588DDAFE4535521CE55
Endpoint long term key pair:
  endpoint public key X: 07B857B9B7F1147E20F4DBE6723CE5F46EF8670CBA20F56297F515C8265E4E42
  endpoint public key Y: 5F1FC9B5DAFB62DAAFB5DC9AA6F8B2EDC1CDD43E20A614EF2F8703FA1459721C
  endpoint private key: 13849B3560961053D985DA5E0FA2AF05EAA99DD73EC50CC4A87029A24D7C331B

>>00A4040005AAAAAAAAAA00

endpoint: received SELECT command

<<5C0201009000

Generate ephemeral keys:
  ephemeral public key X: F98CCA31651AD2E63266144B2450FD6081D8FEA8CEB826E1FB10E8034E932446
  ephemeral public key Y: CAD19D201062DD1C7CB0BB293BF16A4BEFB2ED500977E7197E01F26906E39B5F
  ephemeral private key: B0FBA5FB966FDD3BE4096FA65307AB0A7A3BB914625BBFD3CB57DAD9183E19CB

>>80800000635C020100874104F98CCA31651AD2E63266144B2450FD6081D8FEA8
CEB826E1FB10E8034E932446CAD19D201062DD1C7CB0BB293BF16A4BEFB2ED50
0977E7197E01F26906E39B5F4C10BF1C41268230AF76BFFE3E7C5D00CF4A4D08
888888888888888800

endpoint: received AUTH0 command
Generate ephemeral keys:
  ephemeral public key X: 43D605526999F032E08F314F22EBCE051D1DAE53DC71F1C4D614B0337BB17F20
  ephemeral public key Y: 3F95D4C06AB8966D2B9A0D3C4BC446DB9343EBF27F9EF811F242A37118AD4F10
  ephemeral private key: E585C9EE89075F795452879AC38261ED0667C6396A34914DEE0681E8DC22A182
Key Derivation:
K: ACEDD14246C16AAF4561E177E567192454C06B2AAEEDB7E278980C25DD7994A2
  endpoint ephemeral public key X: 43D605526999F032E08F314F22EBCE051D1DAE53DC71F1C4D614B0337BB17F20
  reader ephemeral public key X: F98CCA31651AD2E63266144B2450FD6081D8FEA8CEB826E1FB10E8034E932446
transaction identifier: BF1C41268230AF76BFFE3E7C5D00CF4A
interface byte: 5E
flag: 0000
HKDF output: 55AC05B56D3081AAEF7BD8EB53F9B5BAAE126251B6224775A72770B3E10BC65D
F663103140A7832C8AD155E7377E63303DE2D8811B60F1225CAD4FCBDE0BAA3C
Kcmac: 55AC05B56D3081AAEF7BD8EB53F9B5BA
Kenc: AE126251B6224775A72770B3E10BC65D
Kmac: F663103140A7832C8AD155E7377E6330
Krmac: 3DE2D8811B60F1225CAD4FCBDE0BAA3C

<<86410443D605526999F032E08F314F22EBCE051D1DAE53DC71F1C4D614B0337B
B17F203F95D4C06AB8966D2B9A0D3C4BC446DB9343EBF27F9EF811F242A37118
AD4F109000

generate authentication hash:
sha256 hash input: 4D088888888888888888888862043D605526999F032E08F314F22EBCE051D1DAE53
DC71F1C4D614B0337BB17F208720F98CCA31651AD2E63266144B2450FD6081D8
```



```
52 FEA8CEB826E1FB10E8034E9324464C10BF1C41268230AF76BFFE3E7C5D00CF4A
53 9304415D9569
54 sha256 hash output: 9A2A933D4B90F5A9CFB0B5524E36B10D3669B91F2526F6C0FC2369BD98A327A0
55
56 >>80810000429E40CCE7447AC8D0112C24AE4A261AF63EBA7B585126FFA4CE4C06
57 1D11D97B98151CB7D85BDCCA539D152B544B97647DD5CD38DCBDBD82EF93F5B5
58 796FFF3C2C0FD700
59
60 endpoint: received AUTH1 command
61 generate authentication hash:
62 sha256 hash input: 4D08888888888888888888888888888862043D605526999F032E08F314F22EBCE051D1DAE53
63 DC71F1C4D614B0337BB17F208720F98CCA31651AD2E63266144B2450FD6081D8
64 FEA8CEB826E1FB10E8034E9324464C10BF1C41268230AF76BFFE3E7C5D00CF4A
65 9304415D9569
66 sha256 hash output: 9A2A933D4B90F5A9CFB0B5524E36B10D3669B91F2526F6C0FC2369BD98A327A0
67 endpoint: successful signature verification
68 Derive Kdh:
69 ephemeral public key X: F98CCA31651AD2E63266144B2450FD6081D8FEA8CEB826E1FB10E8034E932446
70 ephemeral public key Y: CAD19D201062DD1C7CB0BB293BF16A4BEFB2ED500977E7197E01F26906E39B5F
71 ephemeral private key: E585C9EE89075F795452879AC38261ED0667C6396A34914DEE0681E8DC22A182
72 transaction identifier: BF1C41268230AF76BFFE3E7C5D00CF4A
73 ecdh shared secret: 5A2E155294C7D0074DAE3E133D20A6DA60F0A15A02A3A909268BC15D160323FA
74 sha256 input: 5A2E155294C7D0074DAE3E133D20A6DA60F0A15A02A3A909268BC15D160323FA
75 00000001BF1C41268230AF76BFFE3E7C5D00CF4A
76 derived kdh: 18B0CDC20B916B22D2E5D87FDA544D7BD809D171DA00E103A72DF0FF5E5CD185
77 Key Derivation:
78 K: 18B0CDC20B916B22D2E5D87FDA544D7BD809D171DA00E103A72DF0FF5E5CD185
79 endpoint ephemeral public key X: 43D605526999F032E08F314F22EBCE051D1DAE53DC71F1C4D614B0337BB17F20
80 reader ephemeral public key X: F98CCA31651AD2E63266144B2450FD6081D8FEA8CEB826E1FB10E8034E932446
81 transaction identifier: BF1C41268230AF76BFFE3E7C5D00CF4A
82 interface byte: 5E
83 flag: 0000
84 HKDF output: 65B3C36092CC8B15878DC90E0C3A475D4DC72A2325377760B9B1E1774CBE7ED8
85 46BD16584973BEE37BA5732F3628411B
86 CAC033EE373EBFE533DB574C5C8477A7
87 Kenc: 65B3C36092CC8B15878DC90E0C3A475D
88 Kmac: 4DC72A2325377760B9B1E1774CBE7ED8
89 Krmac: 46BD16584973BEE37BA5732F3628411B
90 Key Derivation:
91 K: 18B0CDC20B916B22D2E5D87FDA544D7BD809D171DA00E103A72DF0FF5E5CD185
92 endpoint ephemeral public key X: 43D605526999F032E08F314F22EBCE051D1DAE53DC71F1C4D614B0337BB17F20
93 reader ephemeral public key X: F98CCA31651AD2E63266144B2450FD6081D8FEA8CEB826E1FB10E8034E932446
94 transaction identifier: BF1C41268230AF76BFFE3E7C5D00CF4A
95 interface byte: 5E
96 flag: 0000
97 HKDF output: 0C0E989932DDE515E6D8409A4628DE5650D43135413724FD097EDFC3332CF0AC
98 Kpersistent: 0C0E989932DDE515E6D8409A4628DE5650D43135413724FD097EDFC3332CF0AC
99 generate authentication hash:
100 sha256 hash input: 4D08888888888888888888888888888862043D605526999F032E08F314F22EBCE051D1DAE53
101 DC71F1C4D614B0337BB17F208720F98CCA31651AD2E63266144B2450FD6081D8
102 FEA8CEB826E1FB10E8034E9324464C10BF1C41268230AF76BFFE3E7C5D00CF4A
103 93044E887B4C
104 sha256 hash output: 48BCCE4843E4E87A01AEC830A1AAF6E7D1380D950C468F81BB5AD4CF40705040
105 appending confidential mailbox content: 00000000000000000000000000000000
106 appending private mailbox content: 00000000000000000000000000000000
107 wrap response:
108 plaintext:4E04001122339E407D2E38CB7A01825EF105E5B53166BE8E85D24243A1A5D276
109 7737297E733AD2BA3FECD02D38128B2D42794BC13C03028D58DED27B298115D5
```

Copyright © 2025 Car Connectivity Consortium LLC. All rights reserved.
Confidential

1

3

Confidential

<<5C0201009000

Generate ephemeral keys:

ephemeral public key X: 82BF9E948ECBDD73C10C7EB7D34D5BEB31CF2908B09ADAC701CB4B1F116F5467

ephemeral public key Y: C9187749054455AA1231FA6562D6D4198779FEC2F4F36DB8D9D6EF2082EEA75B

ephemeral private key: E82CED017293885DC5D157A6DAC87013A72B3182F94939BFAA92BB9A367E7966

>>80800100635C02010087410482BF9E948ECBDD73C10C7EB7D34D5BEB31CF2908

B09ADAC701CB4B1F116F5467C9187749054455AA1231FA6562D6D4198779FEC2

F4F36DB8D9D6EF2082EEA75B4C10F92F7260B588238C1E2A4825AD4D7D2E4D08

888888888888888800

endpoint: received AUTH0 command

Generate ephemeral keys:

ephemeral public key X: 0EA56A82A1AD7FC2C739FBB793C0BC3B8935C2ED46B672EFCB98F7DF124FA7FF

ephemeral public key Y: A4155A91F0FCBB007C61E6C574F0F87D3CAF1F41EDE0DF87F43DB664B2C81540

ephemeral private key: BCD0A7ECE3BD6A27A2E0597DAA647028E9058D415921A282C0B18DE90E6EFA94

Key Derivation:

K: B1E9126FBB4FFCA027AE116FC242A1F93093082DE8661B3CD1942078DEB384FD

endpoint ephemeral public key X:

0EA56A82A1AD7FC2C739FBB793C0BC3B8935C2ED46B672EFCB98F7DF124FA7FF

vehicle ephemeral public key X: 82BF9E948ECBDD73C10C7EB7D34D5BEB31CF2908B09ADAC701CB4B1F116F5467

transaction identifier: F92F7260B588238C1E2A4825AD4D7D2E

interface byte: 5E

flag: 0100

HKDF output: 960BA7366865139D9EFAEFB53B8CB18713736C529481B67615811D4EA9F49650

AAA090CC271484E68BB5B6EE18655AAFA5B82D564DCD9961DEE2830C6550E1C1

Kcmac: 960BA7366865139D9EFAEFB53B8CB187

Kenc: 13736C529481B67615811D4EA9F49650

Kmac: AAA090CC271484E68BB5B6EE18655AAF

Krmac: A5B82D564DCD9961DEE2830C6550E1C1

generate cryptogram:

Kcmac: 960BA7366865139D9EFAEFB53B8CB187

endpoint public key X: 07B857B9B7F1147E20F4DBE6723CE5F46EF8670CBA20F56297F515C8265E4E42

vehicle public key X: F47EB42A771052580C086EFDAAA3084AA3FF7A67CE23393A0373C63487DF1A63

vehicle identifier: 8888888888888888

transaction identifier: F92F7260B588238C1E2A4825AD4D7D2E

input of AES-CMAC: 000000000000000000000003200008001F47EB42A771052580C086EFDAAA3084A

A3FF7A67CE23393A0373C63487DF1A6307B857B9B7F1147E20F4DBE6723CE5F4

6EF8670CBA20F56297F515C8265E4E42F92F7260B588238C1E2A4825AD4D7D2E

8888888888888888

cryptogram: E5B79C3D703D1BE1B26C2A999DB2975B

<<8641040EA56A82A1AD7FC2C739FBB793C0BC3B8935C2ED46B672EFCB98F7DF12

4FA7FFA4155A91F0FCBB007C61E6C574F0F87D3CAF1F41EDE0DF87F43DB664B2

C815409D10E5B79C3D703D1BE1B26C2A999DB2975B9000

Key Derivation:

K: B1E9126FBB4FFCA027AE116FC242A1F93093082DE8661B3CD1942078DEB384FD

endpoint ephemeral public key X:

0EA56A82A1AD7FC2C739FBB793C0BC3B8935C2ED46B672EFCB98F7DF124FA7FF

vehicle ephemeral public key X: 82BF9E948ECBDD73C10C7EB7D34D5BEB31CF2908B09ADAC701CB4B1F116F5467

transaction identifier: F92F7260B588238C1E2A4825AD4D7D2E

interface byte: 5E

flag: 0100

HKDF output: 960BA7366865139D9EFAEFB53B8CB18713736C529481B67615811D4EA9F49650

AAA090CC271484E68BB5B6EE18655AAFA5B82D564DCD9961DEE2830C6550E1C1

1

4 *Listing A-3: Standard Transaction Cryptography Flow (with Reader Intent for Fast Transaction)*

Confidential

Copyright © 2025 Car Connectivity Consortium LLC. All rights reserved.
Confidential

Copyright © 2025 Car Connectivity Consortium LLC. All rights reserved.
Confidential


```
209 wrap response:
210 plaintext:05AAAAAAAAA05BBBBBBBBBB
211 ICV:8A8829B8A956A893BE53B4A0E050E8FC
212 cleartext payload:05AAAAAAAAA05BBBBBBBBBB80000000
213 MAC chaining value:8B6EC24F1A0591AAF0C1AB48C49029B5
214 cipher|mac:56C5CA7C53338B7927DA2B6E8113FD0CD5189CB0DF19EDD4B
215
216 <<56C5CA7C53338B7927DA2B6E8113FD0CD5189CB0DF19EDD4B9000
217
218 unwrap response:
219 cipher|mac:56C5CA7C53338B7927DA2B6E8113FD0CD5189CB0DF19EDD4B
220 plaintext:05AAAAAAAAA05BBBBBBBBBB
```

1 A.4. X.509 Schemes

2 A.4.1. X.509 Basic Certificate Schema

3 Listing A-4: X.509 v3 schema

```
1 Certificate ::= SEQUENCE
2 {
3     tbsCertificate      TBSCertificate,
4     signatureAlgorithm  AlgorithmIdentifier,
5     signatureValue      BIT STRING
6 }
7
8 TBSCertificate ::= SEQUENCE
9 {
10     version             [0] EXPLICIT Version DEFAULT v1,
11     serialNumber        CertificateSerialNumber,
12     signature            AlgorithmIdentifier,
13     issuer               Name,
14     validity             Validity,
15     subject              Name,
16     subjectPublicKeyInfo SubjectPublicKeyInfo,
17     issuerUniqueID      [1] IMPLICIT UniqueIdentifier OPTIONAL,
18                         -- If present, version SHALL be v2 or v3
19     subjectUniqueID     [2] IMPLICIT UniqueIdentifier OPTIONAL,
20                         -- If present, version SHALL be v2 or v3
21     extensions          [3] EXPLICIT Extensions OPTIONAL
22                         -- If present, version SHALL be v3
23 }
24
25 Version ::= INTEGER { v1(0), v2(1), v3(2) }
26
27 CertificateSerialNumber ::= INTEGER
28
29 Validity ::= SEQUENCE
30 {
31     notBefore    Time,
32     notAfter     Time
33 }
34
35 Time ::= CHOICE
36 {
37     utcTime        UTCTime,
38     generalTime    GeneralizedTime
39 }
```

```
40
41 UniqueIdentifier ::= BIT STRING
42
43 SubjectPublicKeyInfo ::= SEQUENCE
44 {
45     algorithm      AlgorithmIdentifier,
46     subjectPublicKey BIT STRING
47 }
48
49 Extensions ::= SEQUENCE SIZE (1..MAX) OF Extension
50
51 Extension ::= SEQUENCE
52 {
53     extnID      OBJECT IDENTIFIER,
54     critical    BOOLEAN DEFAULT FALSE,
55     extnValue   OCTET STRING
56               -- contains the DER encoding of an ASN.1 value
57               -- corresponding to the extension type identified
58               -- by extnID
59 }
60
61 AlgorithmIdentifier ::= SEQUENCE
62 {
63     algorithm  OBJECT IDENTIFIER,
64     parameters ANY DEFINED BY algorithm OPTIONAL
65 }
66
67 Name ::= CHOICE { -- only one possibility for now --
68     rdnSequence RDNSequence }
69
70 RDNSequence ::= SEQUENCE OF RelativeDistinguishedName
71
72 RelativeDistinguishedName ::= SET SIZE (1..MAX) OF AttributeTypeAndValue
73
74 AttributeTypeAndValue ::= SEQUENCE
75 {
76     type  AttributeType,
77     value AttributeValue
78 }
79
80 AttributeType ::= OBJECT IDENTIFIER
81
82 AttributeValue ::= ANY -- DEFINED BY AttributeType
```

1

2 A.4.2. X.509 Key Usage Extension

3

Listing A-5: X.509 Key Usage extension

```
1 id-ce-keyUsage OBJECT IDENTIFIER ::= { id-ce 15 }
2
3 KeyUsage ::= BIT STRING
4 {
5     digitalSignature      (0),
6     nonRepudiation        (1), -- recent editions of X.509 have
7                           -- renamed this bit to contentCommitment
8     keyEncipherment       (2),
9     dataEncipherment      (3),
```

```
10 keyAgreement      (4),
11 keyCertSign      (5),
12 cRLSign          (6),
13 encipherOnly     (7),
14 decipherOnly     (8)
15 }
```

1 A.4.3. X.509 Basic Constraints Extension

2 *Listing A-6: X.509 Basic Constraints extension*

```
1 id-ce-basicConstraints OBJECT IDENTIFIER ::= { id-ce 19 }
2
3 BasicConstraints ::= SEQUENCE
4 {
5     CA                BOOLEAN DEFAULT FALSE,
6     pathLenConstraint  INTEGER (0..MAX) OPTIONAL
7 }
```

3 A.4.4. X.509 Authority Key Identifier Extension

4 *Listing A-7: X.509 Authority Key Identifier extension*

```
1 AuthorityKeyIdentifier ::= SEQUENCE
2 {
3     keyIdentifier          [0] KeyIdentifier          OPTIONAL,
4     authorityCertIssuer    [1] GeneralNames            OPTIONAL,
5     authorityCertSerialNumber [2] CertificateSerialNumber OPTIONAL
6 }
7
8 KeyIdentifier ::= OCTET STRING
```

5 A.4.5. X.509 Subject Key Identifier Extension

6 *Listing A-8: X.509 Subject Key Identifier extension*

```
1 SubjectKeyIdentifier ::= KeyIdentifier
2
3 KeyIdentifier ::= OCTET STRING
```

7 A.4.6. X.509 ECDSA Signature

8 *Listing A-9: X.509 ECDSA Signature Format*

```
1 Ecdsa-Sig-Value ::= SEQUENCE
2 {
3     r  INTEGER,
4     s  INTEGER
5 }
```

9 A.4.7. Issuer and Verifier Rules

10 The following rules supersede the issuance and verification rules defined in RFC 5280 [3]. If no
11 specific rules are stated, the RFC 5280 [3] rules shall apply. In the following tables, “defined”
12 values refer to content of Listing 15-5, Listing 15-13, and Listing 15-16.

13 The device OEM Server shall ensure consistency of endpoint issuer CN format and instance
14 CA subject CN format (both either UTF8 or printableString).

- 1 The issuer CN format (either UTF8String or PrintableString) in the external CA certificate shall
- 2 be the same as the subject CN format in the vehicle OEM root certificate.
- 3 The subject CN format (either UTF8String or PrintableString) in the external CA certificate shall
- 4 be the same as the subject CN format in the device OEM root certificate.

Table A-1: Issuer and Verifier Rules Common to External CA, Instance CA and Endpoint

Item	Verifier Rule	Issuer Rule
tbsCertificate.version	Shall reject if not v3.	Shall issue only v3 certificates.
tbsCertificate.serialNumber	May verify only if the verifier has the capability to strictly follow RFC 5280 verification rules.	Shall be compliant with RFC 5280 rules for conforming CAs.
tbsCertificate.signature.algorithm	May verify only if the verifier has the capability to strictly follow RFC 5280 verification rules.	Shall contain defined OID in algorithm attribute. Shall not contain parameters attribute.
tbsCertificate.(issuer subject) rdnSequence path validation and issuer/subject name chaining	May verify only if the verifier has the capability to strictly follow RFC 5280 verification rules.	Shall issue certificates chain with valid issuer/subject name chaining as per RFC 5280.
tbsCertificate.(issuer subject) rdnSequence	Shall accept multiple RelativeDistinguishedName.	Shall contain only 1 RelativeDistinguishedName.
tbsCertificate.(issuer subject) rdnSequence.[].[].type	Shall accept multiple AttributeTypeAndValue per RelativeDistinguishedName.	The RelativeDistinguishedName shall contain only 1 AttributeTypeAndValue.
tbsCertificate.(issuer subject) rdnSe-	May verify only if the verifier has the capability to strictly follow RFC 5280 verification rules.	OID shall be CommonName.
tbsCertificate.(issuer subject) rdnSequence.[].[].value item type	May verify only if the verifier has the capability to strictly follow RFC 5280 verification rules.	Shall be PrintableString or UTF8String.
tbsCertificate.(issuer subject) rdnSequence.[].[].value item type	May verify only if the verifier has the capability to strictly follow RFC 5280 verification rules.	Item type for issuer field in child certificate shall be the same type as subject of parent certificate.
tbsCertificate.(issuer subject) rdnSequence.[].[].value item content	May verify only if the verifier has the capability to strictly follow RFC 5280 verification rules.	Content length shall not exceed 30 bytes.
tbsCertificate.validitytime item type	Shall accept UTCTime or GeneralizedTime time types independently of the represented date. The expiration of the endpoint certificate shall not be checked.	Shall be compliant with RFC 5280 rules for conforming CAs.
tbsCertificate.validitytime item content	Should not verify unless verifier has a trusted source of time. If tag DB _h was present and set to a non-zero value in owner trackKey request (see Section 6.3.4.4), Instance CA needs to be verified if the verifier is the vehicle OEM Server. Where certificate must be issued (not_before field of the certificate) within value of DB _h in hours before current verifier time. For example, if DB _h had a value 48, then	Shall be compliant with RFC 5280 rules for conforming CAs.

Item	Verifier Rule	Issuer Rule
	the certificate must be issued within 48 hours of the current verifier time.	
tbsCertificate.SubjectPublicKeyInfo.algorithm	May verify only if the verifier has the capability to strictly follow RFC 5280 verification rules.	Shall contain defined OID in algorithm attribute. Shall contain defined OID in parameters attribute.
signatureAlgorithm.algorithm	May verify only if the verifier has the capability to strictly follow RFC 5280 verification rules.	Shall contain defined OID in algorithm attribute. Shall not contain parameters attribute.
signatureValue	Shall be verified as per RFC 5280 rules.	Shall be compliant with RFC5280 rules.

1 *Table A-2: Issuer and Verifier Rules For Extensions Common to External CA, Instance CA and Endpoint*

Item	Verifier Rule	Issuer Rule
tbsCertificate.extensions handling of unrecognized extensions	Unrecognized extensions shall be handled as per RFC 5280 rules.	Extensions not listed in this specification shall be marked as non-critical.
tbsCertificate.extensions OIDs of unrecognized extensions	Shall reject if OIDs are not compliant with RFC 5280 OID format rules.	Shall only issue OIDs compliant with RFC 5280 format rules.
tbsCertificate.extensions handling of duplicate extensions	Shall reject if 2 extensions or more have the same OID.	Shall not issue with 2 or more extensions having the same OID.
tbsCertificate.extensions handling of extension order	Shall accept any extension order.	Extensions can be ordered arbitrarily.
tbsCertificate.extensions.[].Authority-KeyIdentifier	Shall reject if AuthorityKeyIdentifier extension is not present. Shall accept if extension is marked as critical.	Shall contain a AuthorityKeyIdentifier extension with the defined extnValue. Shall be marked as non-critical.
tbsCertificate.extensions.[].KeyUsage	Shall reject if KeyUsage extension is not present. Shall reject if extnValue is different from the defined one. Shall reject if extension is not marked as critical.	Shall contain a KeyUsage extension with the defined extnValue. Shall be marked as critical.

2
3 *Table A-3: Issuer and Verifier Rules For Extensions Specific to External CA*

Item	Verifier Rule	Issuer Rule
tbsCertificate.extensions.[].externalCA	Shall reject if extension is not present. Shall reject if extnValue format or content is different from the defined one. Shall reject if extension is not marked as critical.	Shall contain an extension with the defined OID. Shall contain an extension with defined format and content. Shall be marked as critical.
tbsCertificate.extensions.[].SubjectKey-Identifier	Shall reject if SubjectKeyIdentifier extension is not present. Shall also accept if the extension is marked as critical.	Shall contain a SubjectKeyIdentifier extension with the defined extnValue. Shall be marked as non-critical.
tbsCertificate.extensions.[].Basic-Constraints	Shall reject if BasicConstraints extension is not present. Shall reject if extnValue format or content is not the defined one. Shall reject if extension is not marked as critical.	Shall contain a BasicConstraints extension. The extnValue attribute shall comply with the defined format and content. Shall be marked as critical.

4

Table A-4: Issuer and Verifier Rules For Extensions Specific to Instance CA

Item	Verifier Rule	Issuer Rule
tbsCertificate.extensions[].instanceCA	Shall reject if extension is not present. Shall reject if extnValue format or content is different from the defined one. Shall reject if the extension is not marked as critical.	Shall contain an extension with the defined OID. Shall contain an extension with defined format and content. Shall be marked as critical.
tbsCertificate.extensions[].SubjectKey-Identifier	Shall reject if SubjectKeyIdentifier extension is not present. Shall accept if the extension is marked as critical.	Shall contain a SubjectKeyIdentifier extension with the defined extnValue. Shall be marked as non-critical.
tbsCertificate.extensions[].Basic-Constraints	Shall reject if BasicConstraints extension is not present. Shall reject if extnValue format or content is not the defined one. Shall reject if the extension is not marked as critical.	Shall contain a BasicConstraints extension. The extnValue attribute shall comply with the defined format and content. Shall be marked as critical.

Table A-5: Issuer and Verifier Rules For Extensions Specific to Endpoint

Item	Verifier Rule	Issuer Rule
tbsCertificate.extensions[].endpoint	Shall reject if extension is not present. Shall reject if extnValue format or content is different from the defined one. Shall reject if the extension is not marked as critical.	Shall contain an extension with the defined OID. Shall contain an extension with defined format and content. Shall be marked as critical.
tbsCertificate.extensions[].Basic-Constraints	Shall accept if BasicConstraints extension is absent. Shall reject if extnValue format or content is not the defined one. Shall reject if the extension is not marked as critical.	Shall contain a BasicConstraints extension. The extnValue attribute shall comply with the defined format and content. Shall be marked as critical.

Vehicle Public Key Certificate [K], intermediate CA, Vehicle OEM CA Certificate (signed by Device OEM) [M], and vehicle OEM CA (sub)root certificates shall follow the same rules as the external CA certificate with the following exceptions:

For the Vehicle OEM Root CA certificate:

Table A-6: Issuer and Verifier Rules for Vehicle OEM Root CA certificate

Item	Verifier Rule	Issuer Rule
tbsCertificate.extensions[].-AuthorityKeyIdentifier	Shall not verify presence or content of this field.	May contain a AuthorityKeyIdentifier extension with the defined extnValue. If present it shall be marked as non-critical.
tbsCertificate.extensions[].Key-Usage	Shall not verify presence or content of this field.	May contain a KeyUsage extension with the defined extnValue. If present it shall be marked as critical.

For the Vehicle Public Key Certificate

Table A-7: Issuer and Verifier Rules for Vehicle Public Key Certificate

Item	Verifier Rule	Issuer Rule
tbsCertificate.(issuer subject) rdnSequence.[.].value item content	May verify only if the verifier has the capability to strictly follow RFC 5280 verification rules.	Content length shall not exceed 32 bytes.
tbsCertificate.extensions.[.].SubjectKeyIdentifier	Shall accept if SubjectKeyIdentifier extension is not present. If the extension is present, it shall also be accepted if marked as critical.	Should contain a SubjectKeyIdentifier extension with the defined extnValue. If present, it shall be marked as non-critical.

Appendix B.

B.1. Instance Configurations

instance AID = See Section [15.3.2.1](#)

maximum number of endpoints = <Device OEM policy applies>

maximum number of Instance CAs = <Device OEM policy applies>

SHARING_PASSWORD_LENGTH = <vehicle OEM policy, shall be of length ≥ 4 and ≤ 12 >

The vehicle OEM should choose a suitable combination of password length and allowed number of attempts to enter the password in the vehicle UI, following the recommendations in [\[39\]](#).

B.2. Certificate Field Formats

All certificates, except [C] and [N] (which are attestations) as specified in Section [16](#), shall comply with X.509 formatting as per RFC 5280 [\[3\]](#).

All certificates shall use 256-bit ECC public keys.

All certificates shall append the server environment indicator to their subject CN ("-P", "-E", see below) in addition to specific naming requirements for the subject CN as listed below.

B.2.1. Key Identifier Calculation

The key identifier corresponds to the field Subject Key Identifier in Section 4.2.1.2, method (1) of RFC 5280 [\[3\]](#).

B.2.2. OID Assignment

Each certificate shall be identifiable by the following OID values:

Vehicle Public Key Certificates (leaf) [K]

1.3.6.1.4.1.41577.5.1

External CA Certificates (intermediate)[F]

1.3.6.1.4.1.41577.5.2

Instance CA Certificates (intermediate) [E]

1.3.6.1.4.1.41577.5.3

Endpoint Certificates (leaf)[H]

- 1.3.6.1.4.1.41577.5.4
- Vehicle OEM Encryption Certificate (VehicleOEM.Enc.Cert)
- 1.3.6.1.4.1.41577.5.5
- Vehicle OEM Signature Verification Certificate (VehicleOEM.Sig.Cert)
- 1.3.6.1.4.1.41577.5.6
- Device.Enc.Cert (optional as only the PK is sent in the trackKey request)
- 1.3.6.1.4.1.41577.5.7
- Vehicle Intermediate Certificate
- 1.3.6.1.4.1.41577.5.8
- Vehicle OEM CA Certificate [J]
- 1.3.6.1.4.1.41577.5.9
- Vehicle OEM CA Certificate (Signed by Device OEM CA) [M] (optional as only used by some implementation)
- 1.3.6.1.4.1.41577.5.10
- Certification Body Certificate [R]
- 1.3.6.1.4.1.41577.5.14
- SBxD/KIS Intermediate CA Certificate [S] - optional
- 1.3.6.1.4.1.41577.5.15
- SBxD/KIS Endpoint Certificate [T]
- 1.3.6.1.4.1.41577.5.16
- SBxD/KIS Root CA Certificate [U]
- 1.3.6.1.4.1.41577.5.17

Extensions with the above OIDs shall be marked as critical.

B.2.3. Endpoint Certificate [H]

The **endpoint certificate** shall be formatted as follows:

Subject (CN):

DIGK.[OWNER, FRND].[identifier (4-byte printable ASCII)]|[optional: free text]

The subject field holds the endpoint_identifier value provided at Digital Key creation. The endpoint_identifier and thus the subject shall be unique per Digital Key applet instance. The 4-byte identifier is chosen by the device to ensure uniqueness. The free text may be added for debugging or other identification purposes. It shall not contain dots (“.”) or equal-to (“=”) signs. The subject field free text is separated by an equal to [=] sign from the identifier. The equal-to (“=”) delimiter shall only be included if the optional free text is used.

Examples:

DIGK.OWNER.63K7=JOHN_APPLESEED

DIGK.FRND.Nd75=75924635

The issuer field equals the subject of Instance CA Certificate. The allowed character set per element is A–Z, a–z, 0–9, coded in ASCII.

B.2.4. Instance CA Certificate [E]

The **Instance CA Certificate** shall be formatted as follows:

Subject (CN):

<Vehicle OEM Identifier>

The vehicle OEM identifier is defined in [35]. It shall have a length of 4 characters.

The issuer CN of the Instance CA certificate shall be equal to the subject CN of the external Certificate. The format of text used in the issuer CN of the Instance CA certificate (either PrintableString or UTF8String) shall also match the format of the text used in the subject CN of the external Certificate.

B.2.5. External CA Certificate [F]

The external CA certificate issuer CN equals the subject CN of the vehicle OEM root certificate and the subject CN equals the subject CN of the device OEM root certificate. The format of the subject CN in the vehicle OEM root certificate and the device OEM root certificate shall follow the same rules as outlined in section A.4.7.

The CN fields for subject and issuer in the external CA certificate shall not exceed 30 bytes in DER coding.

B.2.6. Vehicle Public Key/Leaf Certificate [K]

The **Vehicle Public Key Certificate** [K] shall be formatted as follows:

The subject field of [K] shall contain the following information as a readable string:

V.[vehicle_oem].[datacenter/region].[brand].[vehicle_identifier]

All elements are separated by a “.” (dot).

The allowed character set per element is A–Z, a–z, 0–9, coded in ASCII.

[vehicle_oem] defines the Instance CA assignment. It shall match the Vehicle OEM Identifier in Table 2-1 in [35].

Datacenter/region field contains information required by the server, e.g., to know which endpoint/region and server environment to connect to in order to reach the vehicle. It shall be 3 bytes.

It is strongly recommended to have one worldwide endpoint per server environment with the following definition:

- WWP: region=WorldWide, environment=PROD
- WWE: region=WorldWide, environment=E2E (Testing)

If required, possible regional values are:

- EUE: region=EUR, environment=E2E
- EUP: region=EUR, environment=PROD
- USE: region=USA, environment=E2E
- USP: region=USA, environment=PROD
- CHE: region=CHN, environment=E2E
- CHP: region=CHN, environment=PROD

The [brand] field represents the Vehicle Brand Name and should have a granularity that is used to identify the app to be launched for this vehicle. It shall be 4 bytes.

The [vehicle_identifier] is used to associate vehicle and certificate, without using the VIN. The vehicle_identifier requires 16 bytes of ASCII to represent the 8-byte value.

The following part of the subject field is used as ROUTING_INFORMATION which is needed by Owner Device (provided by Vehicle during Owner Pairing) and by Receiver device (provided by Owner Device during key sharing) to determine the vehicle OEM server instance to connect to:

vehicle_oem.datacenter/region.brand

The vehicle_identifier is not part of the routing information.

B.2.7. Intermediate Vehicle Certificate

An intermediate certificate may be used to attest to the vehicle public key. The Vehicle OEM CA attests to the intermediate certificate:

Vehicle OEM CA —> Intermediate Vehicle Certificate —> Vehicle Public Key

Subject CN is defined by the vehicle OEM. It shall terminate with an indication of the environment in which the certificate is valid.

Subject (CN):

<Vehicle OEM defined>-INTERMEDIATE-<region>-<environment>

Example:

Vehicle OEM CA-INTERMEDIATE-WW-P

B.2.8. Vehicle OEM CA Certificate [J]

The Vehicle OEM CA certificate is trusted by an offline process by each device OEM. It is also embedded in the vehicle.

Issuer CN and subject CN are defined by the vehicle OEM. Both shall terminate with an indication of the environment in which the certificate is valid.

Subject (CN):

< Vehicle OEM defined>-<environment>

Example:

Vehicle OEM CA-P

B.2.9. Vehicle OEM CA Certificate (signed by Device OEM) [M]

The Vehicle OEM CA certificate (signed by Device OEM) issuer CN equals subject CN of the Device OEM root certificate[D] and the subject CN equals the subject CN of the vehicle OEM root certificate [J]. The CN field for subject and issuer in the Vehicle OEM CA certificate (signed by Device) shall not exceed 30 bytes in DER coding.

B.2.10. Vehicle OEM Privacy Encryption Certificate

The Vehicle OEM privacy encryption certificate shall be formatted as follows:

Subject (CN):

<Vehicle OEM defined>-ENC-<region>-<environment>

Issuer:

Vehicle OEM-ENC-WW-P

B.2.11. Vehicle OEM Signature Verification Certificate

The Vehicle OEM signature certificate should be formatted as follows:

Subject (CN):

<Vehicle OEM defined>-TERM-<region>-<environment>

Example:

Vehicle OEM-TERM-WW-P

B.2.12. SBxD/KIS Server Endpoint Certificate

The SBxD/KIS server endpoint certificate extension with the OID 1.3.6.1.4.1.41577.5.16 shall be formatted as follows:

Key_server_provider_identifier

- If key server is SBOD then the identifier shall have the format “SBOD-xxxx-yyy-zzzz”
- If key server is SBFD then the identifier shall have the format “SBFD-xxxx-yyy-zzzz”
- If key server is KIS then the identifier shall have the format “KIS-xxxx-yyy-zzzz”

where,

xxxx = This is the identifier of the entity operating the key server. It shall match the CCC-registered OEM Identifier in [\[35\]](#).

yyy = region+environment (e.g. “USP”), with two letters for the region and one for the server environment.

zzzz = sub-server identifier, to be assigned freely by the server provider for more fine-grained identification.

Service_provider_identifier

Not defined by CCC.

Service_identifier

Not defined by CCC.

B.2.13. Validity Period

The fields “not before” and “not after” of the certificates shall be set to the following values:

External CA Certificate, issued by Vehicle OEM

not before = date and time of certificate creation

not after = <Vehicle OEM policy applies>

Instance CA Certificate [E]

not before = date and time of certificate creation

not after = <device and vehicle OEM policies apply>
The expiration of the Instance CA Certificate shall be checked by the vehicle, the owner device, and the key tracking server.

Endpoint Certificate

not before = issuance date and time
not after = 99991231235959Z (as defined in [3])
The expiration date of the endpoint certificate should not be checked anywhere in the system.
Note: The validity definition in the receiver's attestation package determines the lifetime of the key in the system.

Vehicle OEM Privacy Encryption Certificate

not before = issuance date
not after = <vehicle OEM policy applies>

Vehicle OEM Signature Certificate

not before = issuance date
not after = <vehicle OEM policy applies>
The undefined expiration date value 99991231235959Z shall be accepted by the framework and applet implementation.

Appendix C.

C.1. External CA Certificate

The following examples of certificate for illustration purpose using a “mock” environment (“-M”). The certificates may vary within the RFC5280 limits, depending on vehicle OEM issuance policies.

```
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
      42:76:cb:ad:b3:1a:8c:4a:72:e0:1c:5a:5c:ce:59:6f:67:42:d9:c5
    Signature Algorithm: ecdsa-with-SHA256
    Issuer: CN=MOCK-CAR-ROOT-CA-E
    Validity
      Not Before: Jan  1 00:00:00 2019 GMT
      Not After : Jan  1 00:00:00 2049 GMT
    Subject: CN=Device OEM Root Certificate
    Subject Public Key Info:
```

```
Public Key Algorithm: id-ecPublicKey
Public-Key: (256 bit)
pub:
    04:86:86:92:e0:41:b2:25:58:39:c7:85:62:03:b2:
    d9:ae:fd:28:5c:f9:87:94:ec:72:69:37:cb:58:87:
    63:28:91:d8:99:4a:85:c0:29:9a:d3:2d:f8:09:4d:
    75:55:51:56:25:57:5e:74:1b:0d:82:42:cc:08:b2:
    6a:c7:de:43:5e
ASN1 OID: prime256v1
NIST CURVE: P-256
X509v3 extensions:
    X509v3 Authority Key Identifier:
        key-
id:23:18:E5:56:71:F0:8E:AE:21:21:42:A8:17:72:0F:B8:17:EE:93:BF
    X509v3 Subject Key Identifier:
        65:17:11:D3:7B:96:37:00:21:3F:F2:8C:9C:4F:55:02:E4:D8:51:76
    X509v3 Basic Constraints: critical
        CA:TRUE, pathlen:1
    1.3.6.1.4.1.41577.5.2: critical
        0....
    X509v3 Key Usage: critical
        Certificate Sign
Signature Algorithm: ecdsa-with-SHA256
    30:44:02:20:0c:54:51:3f:a1:cd:04:82:dc:49:13:f5:37:4c:
    fa:5e:7e:4c:40:61:86:55:0b:12:3a:71:93:a1:22:22:2a:69:
    02:20:04:2c:2f:06:65:3d:68:a2:81:cf:65:bc:57:85:b0:ca:
    34:91:8b:51:18:83:59:8b:91:1e:58:89:a2:bd:cd:e1
```

C.2. Vehicle OEM Intermediate Certificate

```
Certificate:
Data:
    Version: 3 (0x2)
    Serial Number:
        6d:da:2b:a0:69:c7:55:5a:db:84:a3:ee:dd:a3:83:fc:8a:33:b3:97
    Signature Algorithm: ecdsa-with-SHA256
    Issuer: CN=MOCK-CAR-ROOT-CA-E
    Validity
```

```
1         Not Before: Jan  1 00:00:00 2019 GMT
2         Not After : Jan  1 00:00:00 2049 GMT
3         Subject: CN=MOCK-CAR-INTERMEDIATE-US-E
4         Subject Public Key Info:
5             Public Key Algorithm: id-ecPublicKey
6                 Public-Key: (256 bit)
7                 pub:
8                     04:84:22:42:f6:18:2b:a1:c1:13:8d:32:b7:7f:b9:
9                     f7:f3:7b:70:03:4b:9f:04:44:3a:5b:ea:3c:18:8b:
10                    ea:db:36:49:0a:7e:95:f9:1a:4c:16:2a:cf:c3:40:
11                    1c:3a:4f:4e:5a:59:25:1d:45:24:3a:c8:54:4a:66:
12                    5c:b9:51:42:2f
13                 ASN1 OID: prime256v1
14                 NIST CURVE: P-256
15         X509v3 extensions:
16             X509v3 Authority Key Identifier:
17                 key-
18 id:23:18:E5:56:71:F0:8E:AE:21:21:42:A8:17:72:0F:B8:17:EE:93:BF
19
20             X509v3 Subject Key Identifier:
21                 E6:97:50:57:10:09:13:C3:AB:07:29:D4:53:3F:4E:11:29:9A:34:83
22             X509v3 Basic Constraints: critical
23                 CA:TRUE, pathlen:0
24                 1.3.6.1.4.1.41577.5.8: critical
25                 0....
26             X509v3 Key Usage: critical
27             Certificate Sign
28         Signature Algorithm: ecdsa-with-SHA256
29                 30:45:02:20:08:b4:7d:06:3f:c8:f0:27:95:5d:5d:99:62:ee:
30                 92:91:5e:79:5d:99:4a:4b:5a:a7:6a:f4:2b:6c:16:1d:8c:85:
31                 02:21:00:d8:34:99:b3:49:3c:5b:ee:55:39:98:0b:64:9f:58:
32                 f1:43:54:e6:04:de:a3:9d:22:b3:cc:01:ad:a7:e2:40:04
```

C.3. Vehicle OEM Key/Leaf Certificate

```
34         Certificate:
35             Data:
36                 Version: 3 (0x2)
37                 Serial Number:
38                     0d:7a:48:40:4c:a2:24:14:45:85:d1:64:51:94:80:7b:ea:69:79:22
```

```
1      Signature Algorithm: ecdsa-with-SHA256
2      Issuer: CN=MOCK-CAR-INTERMEDIATE-US-E
3      Validity
4          Not Before: Jan  1 00:00:00 2019 GMT
5          Not After : Jan  1 00:00:00 2049 GMT
6      Subject: CN=V.MOCK.USE.BRND.1111111111111111
7      Subject Public Key Info:
8          Public Key Algorithm: id-ecPublicKey
9              Public-Key: (256 bit)
10             pub:
11                 04:1c:84:2b:af:21:cc:2b:28:3b:83:03:6b:a5:26:
12                 dd:bc:e9:c2:01:f6:3e:80:dc:73:50:88:c4:f3:ca:
13                 90:ff:f2:a4:08:ba:74:4f:78:b6:68:c8:8b:63:d0:
14                 90:a3:e3:48:d2:4d:29:d1:53:f5:31:15:e0:8a:31:
15                 4b:ff:d6:86:01
16             ASN1 OID: prime256v1
17             NIST CURVE: P-256
18      X509v3 extensions:
19          X509v3 Authority Key Identifier:
20              key-
21      id:E6:97:50:57:10:09:13:C3:AB:07:29:D4:53:3F:4E:11:29:9A:34:83
22
23          X509v3 Subject Key Identifier:
24              57:11:1D:D5:E7:5E:24:75:AB:5F:07:BC:96:FB:69:89:FB:7D:0E:AA
25          X509v3 Basic Constraints: critical
26              CA:FALSE
27              1.3.6.1.4.1.41577.5.1: critical
28              0....
29          X509v3 Key Usage: critical
30              Digital Signature
31      Signature Algorithm: ecdsa-with-SHA256
32          30:44:02:20:48:95:c3:bb:16:d0:32:bb:15:c3:0e:2d:50:22:
33          cd:f8:78:a1:4e:87:6c:6b:08:71:09:12:91:fe:48:ba:cf:12:
34          02:20:43:17:ca:c5:d4:4c:67:53:52:7a:f9:d0:64:cb:fb:83:
35          4e:01:78:a0:b5:53:1f:af:83:3d:38:9b:f9:64:ed:b3
```

C.4. Vehicle OEM Privacy Encryption Certificate

```
37      Certificate:
38      Data:
```

```
1      Version: 3 (0x2)
2      Serial Number:
3          7f:86:4e:b6:84:72:d0:52:26:47:92:9d:1e:ab:ae:e4:ef:74:bd:f9
4      Signature Algorithm: ecdsa-with-SHA256
5      Issuer: CN=MOCK-CAR-ROOT-CA-E
6      Validity
7          Not Before: Jan  1 00:00:00 2019 GMT
8          Not After : Jan  1 00:00:00 2049 GMT
9      Subject: CN=MOCK-CAR-ENC-US-E
10     Subject Public Key Info:
11         Public Key Algorithm: id-ecPublicKey
12         Public-Key: (256 bit)
13         pub:
14             04:2b:1d:13:34:c9:d3:16:1c:ca:d0:f5:b9:22:6b:
15             ca:df:ad:5f:c9:b7:83:b2:7f:4b:51:34:84:b7:76:
16             2e:e6:35:33:4d:a4:dc:9b:c7:54:92:3e:21:cb:7b:
17             22:ae:f7:0b:ca:16:6a:12:f0:35:f3:f2:d5:77:aa:
18             a3:b4:8b:d5:f5
19         ASN1 OID: prime256v1
20         NIST CURVE: P-256
21     X509v3 extensions:
22         X509v3 Authority Key Identifier:
23             key-
24 id:23:18:E5:56:71:F0:8E:AE:21:21:42:A8:17:72:0F:B8:17:EE:93:BF
25
26         X509v3 Subject Key Identifier:
27             C3:6C:36:D4:97:A1:31:5B:BB:F2:17:10:A3:E9:6D:12:4D:8D:07:5C
28         X509v3 Basic Constraints: critical
29             CA:FALSE
30             1.3.6.1.4.1.41577.5.5: critical
31             0....
32         X509v3 Key Usage: critical
33             Key Agreement, Encipher Only
34     Signature Algorithm: ecdsa-with-SHA256
35         30:46:02:21:00:f0:f2:bd:59:75:73:8b:60:3f:a9:a4:2d:7d:
36         70:23:4e:73:5a:63:73:59:25:d4:fe:42:4b:33:68:a4:64:5a:
37         e5:02:21:00:b0:df:b3:40:2f:7e:0f:2b:e3:81:43:f8:c5:b3:
38         b1:56:33:4e:7a:87:30:7c:73:64:4d:ad:03:a6:23:01:87:a0
```


C.5. Vehicle OEM Signature Verification Certificate

Certificate:

Data:

Version: 3 (0x2)

Serial Number:

0b:30:c4:1b:39:98:0a:44:41:4b:36:54:55:4f:fc:b5:28:8f:11:cf

Signature Algorithm: ecdsa-with-SHA256

Issuer: CN=MOCK-CAR-ROOT-CA-E

Validity

Not Before: Jan 1 00:00:00 2019 GMT

Not After : Jan 1 00:00:00 2049 GMT

Subject: CN=MOCK-CAR-TERM-US-E

Subject Public Key Info:

Public Key Algorithm: id-ecPublicKey

Public-Key: (256 bit)

pub:

04:62:9e:51:60:2d:98:4c:e3:ea:77:ae:93:6f:e9:

75:7a:34:a9:16:25:99:81:6a:12:df:17:f4:85:84:

ad:77:ee:81:c7:d4:7f:8e:2d:50:29:27:4e:58:32:

ac:69:0a:52:0d:20:22:fb:84:6a:75:c6:a1:fd:e4:

45:ef:d6:a9:b0

ASN1 OID: prime256v1

NIST CURVE: P-256

X509v3 extensions:

X509v3 Authority Key Identifier:

key-

id:23:18:E5:56:71:F0:8E:AE:21:21:42:A8:17:72:0F:B8:17:EE:93:BF

X509v3 Subject Key Identifier:

29:F8:B9:AE:B8:4A:19:76:CF:90:A6:61:D3:0F:49:24:34:6E:31:EC

X509v3 Basic Constraints: critical

CA:FALSE

1.3.6.1.4.1.41577.5.6: critical

0....

X509v3 Key Usage: critical

Digital Signature

Signature Algorithm: ecdsa-with-SHA256

30:45:02:20:3d:d9:06:f8:f6:c0:be:c1:af:88:3c:7f:97:7d:

ad:7c:ad:ee:e7:62:09:71:7f:8f:4d:7c:cd:1f:19:d1:85:70:

```
02:21:00:89:bc:ad:9a:60:8a:b9:49:85:97:9e:d8:e1:5c:25:
f7:c9:55:82:7e:9a:b7:18:9c:5a:2e:ed:d5:fe:d0:e8:c9
```

Appendix D. Owner Pairing Test Vectors [To be updated]

D.1. Derivation of z_0 , z_1 with *Scrypt*

Computed by the Vehicle OEM Server as well as the device.

Parameters

Password: "pleaseletmein" (ASCII, 13 bytes)

Salt: "yellowsubmarines" (ASCII, 16 bytes)

Nscrypt: 32768 (cost parameter)

r: 8 (block size)

p: 1 (parallelization parameter)

dkLen: 80 (output length)

Results

dk: (full 80 bytes)

6408161bbc64a41827cf072c62efdae535739e51

3ad3050e66a9f53eb69c15bb3c220f0acb5f7f02

dd008ce70d974431369ef8e569dee33977fa9fd2

e13a55c8c80c244a05559264b0498c1f5216f327

z0: (left 40 bytes)

6408161bbc64a41827cf072c62efdae535739e51

3ad3050e66a9f53eb69c15bb3c220f0acb5f7f02

z1: (right 40 bytes)

dd008ce70d974431369ef8e569dee33977fa9fd2

e13a55c8c80c244a05559264b0498c1f5216f327

D.2. Computation of w_0 , w_1

w_0 and w_1 computed per FIPS 186-4, B.5.1 Per-Message Secret Number Generation

Using Extra Random Bits.

$w_0 = (z_0 \bmod (n - 1)) + 1$

$w_1 = (z_1 \bmod (n - 1)) + 1$

With n being the order of base point G as defined for NIST P-256.

Parameters

n: (32 bytes)

ffffffff00000000ffffffffffffffff

bce6faada7179e84f3b9cac2fc632551

z0: (40 bytes)

```
6408161bbc64a41827cf072c62efdae535739e51
3ad3050e66a9f53eb69c15bb3c220f0acb5f7f02
z1: (40 bytes)
dd008ce70d974431369ef8e569dee33977fa9fd2
e13a55c8c80c244a05559264b0498c1f5216f327
```

Results

```
w0: (32 bytes)
e433ab43428320b24fab82f915d1db11 4acd72f8a4bf4fbf3c712b94bcc2f013
w1: (32 bytes)
44363d157f471221b1e75e596ff4714a
712b9578301665d84ec17004952523a8
```

D.3. Computation of L

Only computed on the Vehicle OEM Server.

$L = w1 \times G$

L is the result of scalar multiplication on curve NIST P-256. G is the base point as defined for NIST P-256.

Parameters

```
w1: (32 bytes)
44363d157f471221b1e75e596ff4714a
712b9578301665d84ec17004952523a8
G.x: (32 bytes)
6b17d1f2e12c4247f8bce6e563a440f2 77037d812deb33a0f4a13945d898c296
G.y: (32 bytes)
4fe342e2fe1a7f9b8ee7eb4a7c0f9e16 2bce33576b315ececbb6406837bf51f5
```

Results

```
L.x: (32 bytes)
ff69eb6086938b3cce2c9e64dcacea1a 925918e75e8c17948d316322d370123f
L.y: (32 bytes)
69132aed7398919e6e6614f7627b0a54
060c5a8c0d93d2754166ab10fea6a8ff
```

D.4. Computation of X

Computed by the device.

$X = x \times G + w0 \times M$

x is a random scalar. G is the base point as defined for NIST P-256 and M a fixed point as defined by the spec.

Parameters

```
1      x: (32 bytes)
2      000102030405060708090a0b0c0d0e0f
3      101112131415161718191a1b1c1d1e1f
4      w0: (32 bytes)
5      e433ab43428320b24fab82f915d1db11
6      4acd72f8a4bf4fbf3c712b94bcc2f013
7      G.x: (32 bytes)
8      6b17d1f2e12c4247f8bce6e563a440f2
9      77037d812deb33a0f4a13945d898c296
10     G.y: (32 bytes)
11     4fe342e2fe1a7f9b8ee7eb4a7c0f9e16
12     2bce33576b315ececbb6406837bf51f5
13     M: (65 bytes)
14     04
15     886e2f97ace46e55ba9dd7242579f299
16     3b64e16ef3dcab95afd497333d8fa12f
17     5ff355163e43ce224e0b0e65ff02ac8e
18     5c7be09419c785e0ca547d55a12e2d20
19
20     Results
21     xG.x: (32 bytes)
22     7a593180860c4037c83c12749845c8ee
23     1424dd297fadcb895e358255d2c7d2b2
24     xG.y: (32 bytes)
25     a8ca25580f2626fe579062ff1b99ff91 c24a0da06fb32b5be20148c9249f5650
26     w0M.x: (32 bytes)
27     0d556afe7d4cf1b87a9baa429ab9fb57 5de4f36d412cbd7d0dc095779590f5aa
28     w0M.y: (32 bytes)
29     5f941aae1286700e4dab2ff38b4eeb2d 6ceb5dd3f7072bc84ee74360d0f7ad0c
30     X.x: (32 bytes)
31     f44555207a617fd90900dba5c8e6f81e ddbd87590873a63b9057dda9f138dbc1
32     X.y: (32 bytes)
33     6f453195f6452ce71d399052435952b8
34     9a10b927435574f5e3707eae031c40e0
```

D.5. Computation of Y

Computed by the vehicle.

$$Y = y \times G + w0 \times N$$

y is a random scalar. G is the base point as defined for NIST P-256 and N a fixed point as defined by the spec.

Parameters

y: (32 bytes)

f1e1d1c1b1a191817161514131211101

f0e0d0c0b0a090807060504030201000

w0: (32 bytes)

e433ab43428320b24fab82f915d1db11

4acd72f8a4bf4fbf3c712b94bcc2f013

G.x: (32 bytes)

6b17d1f2e12c4247f8bce6e563a440f2

77037d812deb33a0f4a13945d898c296

G.y: (32 bytes)

4fe342e2fe1a7f9b8ee7eb4a7c0f9e16

2bce33576b315ececbb6406837bf51f5

N: (65 bytes)

04

d8bbd6c639c62937b04d997f38c37707

19c629d7014d49a24b4f98baa1292b49

07d60aa6bfade45008a636337f5168c6

4d9bd36034808cd564490b1e656edbe7

Results

yG.x: (32 bytes)

d05d3f33d7a67710f70275600e2905e4

4f436e3331a1f479f7de0b650f679d12

yG.y: (32 bytes)

1c404aa84e6f75fb75d0526ff48e004b 4ea689d7fc619a05e0ba37dfbef340bf

w0N.x: (32 bytes)

4f8cd0a6e01386dac1b2e1af01e1e56a 65d7beccfcb7ed39437308f4bd99d3b6

w0N.y: (32 bytes)

e89a7da518b37a32f9acfe497da5fd16 05b80820b4ce92406569de793811d938

Y.x: (32 bytes)

b6fdaf3f6949869d68f667108b75e4ce 74847e8953d1e3c6aae21699e8027211

Y.y: (32 bytes)

c2d9b2b2a906cc7ea7020715dec44e95 659e3fc8994f635b95e7c9ea5c362cbe

D.6. Computation of Z, V (by device)

Z, V are keys shared by the device and vehicle.

$T = Y - w0 \times N$

$Z = x \times T$

$V = w1 \times T$

Parameters

x: (32 bytes)

000102030405060708090a0b0c0d0e0f

101112131415161718191a1b1c1d1e1f

w0: (32 bytes)

e433ab43428320b24fab82f915d1db11

4acd72f8a4bf4fbf3c712b94bcc2f013

w1: (32 bytes)

44363d157f471221b1e75e596ff4714a

712b9578301665d84ec17004952523a8

Y: (65 bytes)

04

b6fdaf3f6949869d68f667108b75e4ce

74847e8953d1e3c6aae21699e8027211

c2d9b2b2a906cc7ea7020715dec44e95

659e3fc8994f635b95e7c9ea5c362cbe

N: (65 bytes)

04

d8bbd6c639c62937b04d997f38c37707

19c629d7014d49a24b4f98baa1292b49

07d60aa6bfade45008a636337f5168c6

4d9bd36034808cd564490b1e656edbe7

Results

T.x: (32 bytes)

d05d3f33d7a67710f70275600e2905e4

4f436e3331a1f479f7de0b650f679d12

T.y: (32 bytes)

1c404aa84e6f75fb75d0526ff48e004b

4ea689d7fc619a05e0ba37dfbef340bf

Z.x: (32 bytes)

89af176e8122e67c00dbcea089bc4993 5634132b1b226030d2b14f16b3e73351

Z.y: (32 bytes)

c2254b1b62477fdc976379f5ae7c57c6 ac2b31ef9a032c33e4677c5c3acbb1d3

V.x: (32 bytes)

2f229f13e1ebfb6442a67eebdafeb23b2 f6e656597384035a8a1e50ad95d24211

V.y: (32 bytes)

339e90a669dd8a56fb2524fb6e6c784b 89019c1130c2def98143fb46dcc507d2

D.7. Computation of Z , V (by vehicle)

Z , V are keys shared by the device and vehicle.

$$Z = y \times (X - w0 \times M)$$

$$V = y \times L$$

Parameters

y: (32 bytes)

f1e1d1c1b1a191817161514131211101

f0e0d0c0b0a090807060504030201000

w0: (32 bytes)

e433ab43428320b24fab82f915d1db11

4acd72f8a4bf4fbf3c712b94bcc2f013

x: (65 bytes)

04

f44555207a617fd90900dba5c8e6f81e

ddbd87590873a63b9057dda9f138dbc1

6f453195f6452ce71d399052435952b8

9a10b927435574f5e3707eae031c40e0

M: (65 bytes)

04

886e2f97ace46e55ba9dd7242579f299

3b64e16ef3dcab95afd497333d8fa12f

5ff355163e43ce224e0b0e65ff02ac8e

5c7be09419c785e0ca547d55a12e2d20

L: (65 bytes)

04

ff69eb6086938b3cce2c9e64dcacea1a

925918e75e8c17948d316322d370123f

69132aed7398919e6e6614f7627b0a54

060c5a8c0d93d2754166ab10fea6a8ff

Results

z.x: (32 bytes)

89af176e8122e67c00dbcea089bc4993

5634132b1b226030d2b14f16b3e73351

z.y: (32 bytes)

c2254b1b62477fdc976379f5ae7c57c6

ac2b31ef9a032c33e4677c5c3acbb1d3

v.x: (32 bytes)

2f229f13e1ebfb6442a67eebda fb23b2

```
1          f6e656597384035a8a1e50ad95d24211
2          v.y: (32 bytes)
3          339e90a669dd8a56fb2524fb6e6c784b
4          89019c1130c2def98143fb46dcc507d2
```

5 *D.8. Derivation of K, CK, SK*

6 K is the secret shared by the device and vehicle.

7 $K = \text{SHA-256}(\text{len}(X) \parallel X \parallel \text{len}(Y) \parallel Y \parallel \text{len}(Z) \parallel Z \parallel \text{len}(V) \parallel V \parallel \text{len}(w0) \parallel w0)$

8 $\text{len}(x)$ denotes the length of a string in bytes, represented as an 8-byte little-endian number.

9 **Parameters**

```
10          w0: (32 bytes)
11          e433ab43428320b24fab82f915d1db11
12          4acd72f8a4bf4fbf3c712b94bcc2f013
13          len(w0): (8 bytes)
14          2000000000000000
15          x: (65 bytes)
16          04
17          f44555207a617fd90900dba5c8e6f81e
18          ddbd87590873a63b9057dda9f138dbc1
19          6f453195f6452ce71d399052435952b8
20          9a10b927435574f5e3707eae031c40e0
21          y: (65 bytes)
22          04
23          b6fdaf3f6949869d68f667108b75e4ce
24          74847e8953d1e3c6aae21699e8027211
25          c2d9b2b2a906cc7ea7020715dec44e95
26          659e3fc8994f635b95e7c9ea5c362cbe
27          z: (65 bytes)
28          04
29          89af176e8122e67c00dbcea089bc4993
30          5634132b1b226030d2b14f16b3e73351
31          c2254b1b62477fdc976379f5ae7c57c6
32          ac2b31ef9a032c33e4677c5c3acbb1d3
33          v: (65 bytes)
34          04
35          2f229f13e1ebfb6442a67eebda fb23b2
36          f6e656597384035a8a1e50ad95d24211
37          339e90a669dd8a56fb2524fb6e6c784b
38          89019c1130c2def98143fb46dcc507d2
```



```
1      len(X/Y/Z/V) : (8 bytes)
2      4100000000000000
3
4      Results
5      K: (32 bytes)
6      381fc44894aedc0fd37257fdca763ee4
7      9f695017ba5e1df74a1b8bbf27fe1e0d
8      CK: (16 bytes)
9      381fc44894aedc0fd37257fdca763ee4
10     SK: (16 bytes)
11     9f695017ba5e1df74a1b8bbf27fe1e0d
```

11 *D.9. Derivation of Evidence Keys K1, K2*

12 K1 and K2 are the MAC keys used for key confirmation.
13 HKDF-SHA-256 is used as the KDF, defined by IETF RFC5869.

14 **Parameters**

```
15 L: 32
16 IKM/CK: (16 bytes)
17 381fc44894aedc0fd37257fdca763ee4
18 Info: "ConfirmationKeys" (ASCII, 16 bytes)
19 5b020101 (Agreed SPAKE2+ protocol version, 4 bytes)
20 5c0401010100 (Tag 5C of the SPAKE2+ REQUEST command, 6 bytes)
```

21 **Results**

```
22 K1: (16 bytes)
23 ab667caeffe27505265d0f2026e146ea
24 K2: (16 bytes)
25 af679bf88b734e1cb7cd0243fb21a589
```

26 Note that this calculation is an example of key derivation that uses an undefined protocol version
27 0101. Its intent is to show the key derivation algorithm in principle.

28 *D.10. Computation of Evidences M1, M2*

29 M1 and M2 are the MACs used for key confirmation.
30 $M1 = \text{CMAC}(K1, X)$
31 $M2 = \text{CMAC}(K2, Y)$

32 **Parameters**

```
33 K1: (16 bytes)
34 ab667caeffe27505265d0f2026e146ea
35 K2: (16 bytes)
36 af679bf88b734e1cb7cd0243fb21a589
```

X: (65 bytes)
04
f44555207a617fd90900dba5c8e6f81e
ddb87590873a63b9057dda9f138dbc1
6f453195f6452ce71d399052435952b8
9a10b927435574f5e3707eae031c40e0
Y: (65 bytes)

04
b6fdaf3f6949869d68f667108b75e4ce
74847e8953d1e3c6aae21699e8027211
c2d9b2b2a906cc7ea7020715dec44e95
659e3fc8994f635b95e7c9ea5c362cbe

Results

M1: (16 bytes)
110d49f8c5a896e11d4dde4c3b9704d2
M2: (16 bytes)
23d1a618ad3acbfd7a9bd19fd1737107

D.11. Derivation of System Keys

HKDF-SHA-256 is used as the KDF, defined by IETF RFC 5869.

Parameters

L: 64
IKM/SK: (16 bytes)
9f695017ba5e1df74a1b8bbf27fe1e0d
Info: "SystemKeys" (ASCII, 10 bytes)

Results

Kenc: (16 bytes)
161886cb9ae7403d8dbccfe36b8a0426
Kmac: (16 bytes)
6387ba65479cb7eb9df97bd48ac33159
Krmac: (16 bytes)
41677fb6398459199f1e569760df91c1
LONG_TERM_SHARED_SECRET: (16 bytes)
5c4e19da553524e386faleca91e8ad0e

Appendix E. NFC-F Support [WCC1]

E.1. Introduction

This Appendix defines a protocol to perform the contactless transactions defined in this document over NFC-F.

Support for NFC-F is an optional feature of the Digital Key protocol. The requirements in this appendix apply only if an implementation supports this option.

Implementation may include support for NFC-F to be able to use NFC-F devices for holding Digital Keys or if the contactless transactions defined in this document are used as part of a larger NFC-F transaction.

The protocol in this appendix defines how the device exchanges APDUs with the vehicle over NFC-F. The APDU-based message exchanges defined in this document are not modified in any way when run over NFC-F. Support for NFC-F only affects the contactless interface; it has no impact on the device-internal interfaces to communicate between the Digital Key applet and Digital Key framework or on the communication with the Device OEM Server or Vehicle OEM Server.

The following definitions are specific to this appendix:

- **Protocol Error:** Receiving a response with a wrong syntax or receiving a correct response when it is not expected.
- **Timeout Error:** No response has been received within the defined waiting time.
- **Transmission Error:** Error caused by the receipt of an incorrect frame. This includes the case that the CRC in the NFC-F frame is not correct.

The following abbreviations are specific to this appendix:

RW_ACK	Acknowledge send from vehicle to device
DEVICE_ACK	Acknowledge send from device to vehicle
C-APDU	Command APDU as defined in [1]
fc	Carrier Frequency
NACK	Negative acknowledgement
NFC-F	NFC Type F RF Technology
NFCID2	NFC-F identifier as defined in [DIGITAL].
R-APDU	Response APDU as defined in [1]
RWT	Response waiting time
RWTI	Response waiting time integer

E.2. Device Requirement

To support NFC-F, the following requirements apply on the device side:

- The protocol defined in this appendix shall be implemented by the Digital Key applet.
- The SE shall be compliant with the GlobalPlatform card specification and shall implement GlobalPlatform Contactless Services – Amendment C [21], including support for “Type F”.

- 1 • The Digital Key applet shall register the System Code $ACCC_h$ as part of its contactless
2 protocol parameters for Type F (Sub-tag 80_h of the Type F anti-collision parameters entry tag
3 $A0_h$; see [21]).
- 4 • The device shall configure the NFC controller to route frames containing the NFCID2 of
5 System Code $ACCC_h$ to the SE that hosts the Digital Key applet.
- 6 • The Digital Key applet shall register an NFCID2 (as defined in [DIGITAL]) as part of its
7 contactless protocol parameters for Type F. The NFCID2 is registered as the “PICC
8 Identifier” contained in sub-tag 81_h of the Type F anti-collision parameters entry tag $A0_h$ (see
9 [21]).
- 10 • The following applies only if owner pairing over NFC-F has to be supported:
 - 11 ○ The protocol defined in this appendix shall be implemented by the Digital Key
12 framework
 - 13 ○ The Digital Key framework shall use the System Code $4CCC_h$ for NFC-F.
 - 14 ○ The Digital Key framework shall use an NFCID2 in the range of
15 $02FE000000000000_h$ to $02FEFFFFFFF_h$. The PAD0 field in the SENSEF_RES
16 as defined in [17] shall be configured to be $01FE_h$.
 - 17 ○ The device shall configure the NFC controller to route frames containing the NFCID2
18 of System Code $4CCC_h$ to the host with the Digital Key framework.

19 To support NFC-F, the following requirements apply on the vehicle side:

- 20 • The vehicle shall implement the protocol defined in this appendix. The protocol shall be
21 available for communication on the door NFC readers and console NFC reader.
- 22 • If configured to support NFC-F, the NFC readers shall include polling for NFC-F. The
23 polling shall use the system code for the Digital Key applet except in case of owner pairing,
24 in which case the system code for the Digital Key framework shall be used.

25 *Note:* As only devices that support the requested system code will respond to the polling, there is
26 no risk that the NFC link is established using NFC-F technology without the Digital Key
27 protocol being available.

28 On either side, the APDUs exchanged as part of the Digital Key protocol shall be encapsulated
29 as defined in the following sections before sending and shall be extracted accordingly after
30 receiving.

31 *E.3. Preconditions*

32 This specification is based on NFC-F technology, more specifically on the Type 3 Tag Platform
33 as defined in [17] and [18].

34 The vehicle and the device shall be able to send and receive NFC-F frames as defined in [17].

35 The device shall be able to respond to a SENSEF_REQ command as defined in [17].

36 *E.4. Commands and responses*

37 The formats for NFC-F commands and responses defined in this section shall be transported
38 inside the Payload field of the NFC-F Data and Payload format as defined in [17].

The LEN byte and the CRC are not included in these format definitions. The LEN field has to be prepended and the CRC to be appended to obtain the content for the data field of an NFC-F frame.

The LEN field is supposed to be part of the command or response and therefore constitutes the first byte of each command and response.

E.4.1. Encapsulation Command (ENCAPSULATION_CMD)

The ENCAPSULATION_CMD is used to transfer APDU data from the vehicle to the device.

The ENCAPSULATION_CMD shall be formatted as defined in Table E-1.

Table E-1: ENCAPSULATION_CMD format

Byte 1	Byte 2	Bytes 3–10	Bytes 11–n
C2 _h	P1	NFCID2	PAYLOAD

P1

The P1 field shall be formatted as defined in Table E-2.

Table E-2: Format of the P1 field

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
Message Type				Message Features				
0	0	0	0	0	0	0		CAPDU_DATA
							x	Chaining, if set to 1 _b
0	0	0	1	0	0	0	0	RW_ACK
0	0	1	0	0	0	0	0	NACK
Any other value								RFU

In the following sections, the name in the “Meaning” column is often used to refer to an ENCAPSULATION_CMD with a specific “Message Type” value.

Chaining set to 1_b indicates that the Payload field contains a segment of a larger data. If chaining is set to 1_b, the Command is referred to as “CAPDU_DATA indicating chaining.” If “indicating chaining” is not explicitly mentioned, the bit is set to 0_b.

The device shall ignore commands that have an RFU value.

NFCID2

The vehicle shall set the NFCID2 to the value included in the SENSF_RES response of the targeted device. The device shall compare the NFCID2 contained in this command with its own NFCID2. If they are not equal, the device shall not respond to this command.

PAYLOAD

For RW_ACK and NACK messages, the Payload field shall be empty.

The content of the Payload field for CAPDU_DATA is specified in Appendix [E-5](#).

E.4.2. Encapsulation Response (ENCAPSULATION_RSP)

- 1 The ENCAPSULATION_RSP is used to transfer APDU data from the device to the vehicle.
2 The ENCAPSULATION_RSP shall be formatted as defined in Table E-3.

Table E-3: ENCAPSULATION_RSP format

Byte 1	Byte 2	Bytes 3–10	Bytes 11–n
C3 _h	P1	NFCID2	PAYLOAD

P1

- 5 The P1 field shall be formatted as defined in Table E-4.

Table E-4: Format of the P1 field

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
Message Type				Message Features				
0	0	0	0	0	0	0		RAPDU_DATA
							x	Chaining, if set to 1 _b
0	0	0	1	0	0	0	0	DEVICE_ACK
Any other value								RFU

- 7 In the following sections, the name in the “Meaning” column is often used to refer to an
8 ENCAPSULATION_RSP with a specific “Message Type” value.

9 Chaining set to 1_b indicates that the Payload field contains a segment of a larger data. If chaining
10 is set to 1_b, the response is referred to as “RAPDU_DATA indicating chaining.”. If “indicating
11 chaining” is not explicitly mentioned, the bit is set to 0_b.

NFCID2

13 The device shall set the NFCID2 to its own NFCID2. The vehicle shall verify that the NFCID2
14 in the response is the same as the one sent in the previous command. If the NFCID2 is different,
15 the vehicle shall ignore the response.

PAYLOAD

17 For DEVICE_ACK messages, the Payload field shall be empty.

18 The content of the Payload field for RAPDU_DATA is specified in Appendix [E.5](#).

E.5. Protocol Operation

E.5.1. Protocol activation

21 The protocol is implicitly activated if the device responds to a SENSEF_REQ with a SENSEF_RES
22 that includes an NFCID2 that is associated with one of the system codes defined in Appendix [E-](#)
23 [2](#) (see [\[18\]](#) for more information on activation of a Type 3 Tag Platform and data exchange with
24 a Type 3 Tag Platform).

25 The vehicle shall ignore the values of PAD1, MRTI_{CHECK}, and MRTI_{UPDATE} as defined in [\[17\]](#).

E.5.2. General rules

After sending an `ENCAPSULATION_CMD`, the vehicle shall be ready to receive an `ENCAPSULATION_RSP`. After receiving an `ENCAPSULATION_CMD` without error, the device shall respond with an `ENCAPSULATION_RSP`.

If the length of the C-APDU is 244 bytes or less, the C-APDU shall be transported in the Payload field of a `CAPDU_DATA` message as defined in Appendix [E.4.1](#).

If the length of the R-APDU is 244 bytes or less, the R-APDU shall be transported in the Payload field of an `RAPDU_DATA` message as defined in Appendix [E.4.2](#).

For APDUs with a larger size, refer to Appendix [E.5.3](#).

E.5.3. Chaining

The chaining feature allows transmitting APDUs that do not fit in a single command or response by dividing it into segments. Each segment is transmitted in a separate command or response.

C-APDU

If the length of the C-APDU is above 244 bytes, the C-APDU shall be segmented and transferred in multiple `CAPDU_DATA` messages.

All but the last segment shall be transported in the Payload field of a `CAPDU_DATA` message indicating chaining as defined in Appendix [E.4.1](#). C-APDU segments that are transported in a `CAPDU_DATA` message indicating chaining shall have a Payload field with a size of 244 bytes. The last segment shall be transported in the Payload field of a `CAPDU_DATA` message indicating no chaining.

The vehicle shall send the first segment of the C-APDU in a `CAPDU_DATA` message indicating chaining. After receiving a `CAPDU_DATA` message indicating chaining, the device shall acknowledge the receipt of the segment by sending a `DEVICE_ACK` as defined in Appendix [E.4.2](#). After receiving a `DEVICE_ACK`, the vehicle shall send the next segment in a `CAPDU_DATA` message indicating chaining, or, if it is the last segment, in a `CAPDU_DATA` message indicating no chaining.

R-APDU

If the length of the R-APDU is greater than 244 bytes, the R-APDU shall be segmented and transferred in multiple `RAPDU_DATA` messages.

All but the last segment shall be transported in the Payload field of an `RAPDU_DATA` message indicating chaining as defined in Appendix [E.4.2](#). R-APDU segments that are transported in an `RAPDU_DATA` message indicating chaining shall have a Payload field with a size of 244 bytes. The last segment shall be transported in the Payload field of an `RAPDU_DATA` message indicating no chaining.

The device shall send the first segment of the R-APDU in an `RAPDU_DATA` message indicating chaining. After receiving an `RAPDU_DATA` message indicating chaining, the vehicle shall acknowledge receipt of the segment by sending a `RW_ACK` as defined in Appendix [E.4.1](#). After receiving a `RW_ACK`, the device shall send the next segment in an `RAPDU_DATA` message indicating chaining, or, if it is the last segment, in an `RAPDU_DATA` message indicating no chaining.

E.5.4. Timing requirements

The waiting time used by the protocol is announced in the SENSEF_RES. The protocol uses one waiting time, referred to as RWT.

The SENSEF_RES as defined in [17] has the following format:

Table E-5: SENSEF_RES format

Byte 1	Bytes 2–9	Bytes 10–11	Bytes 12–14	Byte 15	Byte 16	Byte 17	Bytes 18–19
01 _h	NFCID2	PAD0	Undefined Values (PAD1)	MRTI _{CHECK}	MRTI _{UPDATE}	Undefined Values (PAD2)	[RD]

In this specification, Byte 17 (PAD2) is referred to as RWTI. The device shall use a value for RWTI that reflects the maximum time it requires to respond to a CAPDU_DATA message.

The Digital Key applet shall register its RWTI as part of its contactless protocol parameters for Type F. The RWTI corresponds to the last byte of the “Response Time Descriptor” contained in Sub-tag 81_h of the Type F anti-collision parameters entry tag A0_h (see [21]). The recommended value for the first two bytes of the “Response Time Descriptor” (PAD0 of the SENSEF_RES) is 01FF_h.

The format of the RWTI byte shall be as defined in Table E-6:

Table E-6: Format of RWTI

b7	b6	b5	b4	b3	b2	b1	b0	Meaning
					x	x	x	Parameter A
		x	x	x				Parameter B
x	x							Parameter E

The RWT shall be calculated according to the following formula:

$$RWT = T \times ((A+1) + 8 \times (B+1)) \times 4^E$$

where T is a fixed duration with the following value: $T = 256 \times 16/f_c$ (~0.3020 ms).

After receiving the last byte of a ENCAPSULATION_CMD message, the device shall wait no longer than RWT before sending the first byte of an ENCAPSULATION_RSP (which is the LEN byte). If the vehicle does not receive the first byte of an ENCAPSULATION_RSP within $RWT + \Delta RWT$, the vehicle shall treat this as a timeout error.

The value of ΔRWT shall be 20 ms.

E.5.5. Error handling

Device

The device shall not attempt any error recovery. The device shall stay in receive mode when it detects an error.

Vehicle

If a timeout error occurs, the vehicle shall resend the last command.

If the vehicle can detect transmission errors and if it receives a response with a transmission error, the vehicle shall send a NACK as defined in Appendix E.4.1.

After receiving a NACK, the device shall resend the last `ENCAPSULATION_RSP`.
The vehicle shall attempt sequential error recovery (without receiving a valid response in between) up to a maximum of two times.
If not specified otherwise, the handling of protocol errors by the vehicle is implementation-specific.

E.5.6. Protocol deactivation

The vehicle deactivates the protocol either by selecting another System Code via a `SENSF_REQ` or by switching off the operating field.

Appendix F. Digital Key Framework API

This is a normative appendix, specifying API requirements.

F.1. Overview

The API requirements specified in this document enable mobile applications to access the Digital Key framework running the Android system in mobile devices. This specification provides interface definitions for an application developer to allow implementation on Android mobile platforms. The Android APIs are defined in Android Digital Key Framework API [\[29\]](#).

F.2. Functional Requirements

The Digital Key framework shall provide common Digital Key functionality via a set of OS-specific APIs for Vehicle OEM apps. The following high-level services shall be provided to the Vehicle OEM app:

Owner pairing, key sharing, and key management. At a minimum, these APIs shall be able to be used to, respectively:

- Trigger owner pairing
- Trigger key sharing
- List Digital Keys

Additionally, the API should provide interfaces to enable the following functionalities:

- Owner pairing:
 - Trigger owner pairing and provide owner pairing password
 - Request/receive owner pairing status updates
- Key sharing:
 - For owner: Provide entitlements (including proprietary ones), friendly name, and retrieve sharing URL and sharing password from the framework.
 - For receiver: Receive/send sharing URL and capture acceptance/consent.
 - Request/receive receiver key status updates.
 - Send notification to Vehicle OEM app when a receiver key is created on receiver device.
- Key management:
 - Terminate Digital Key

- List all owner/shared Digital Keys with status, entitlements, friendly names, vehicle identifier (vehicleIdentifier), and Digital Key identifier (keyID).
- Deeplink to owner/shared keys to jump from Vehicle OEM app directly to the key in the device native representation.
- Sign vehicle identifier (vehicleIdentifier) and Digital Key identifier (keyID) with attestable key from Digital Key applet instance on receiver device to verify the identifiers in Vehicle OEM Server.
- Sign data with attestable key from Digital Key applet instance on owner or receiver device to verify signature in Vehicle OEM Server.
- Read/write information related to Digital Key (e.g., information stored in private mailbox and Instance CA ID).
- RKE Functionality
 - Trigger event based RKE action
 - Trigger enduring RKE action
 - Callback providing the arbitrary data from request continuation confirmation while action endures
 - Sending of arbitrary data in confirm continuation message as answer to the request above
- Vehicle Management
 - Retrieve vehicle state
 - subscribing to a certain function id range
 - requesting a certain function status value from the vehicle
 - reading the stored status data.

Appendix G.

G.1. Ranging Session Parameter Tables [WCC3]

As part of the ranging session configuration, each ranging session is configured with the following values:

- $N_{\text{RAN}}^k \hat{=} \mathbb{N}^+$, set of all positive natural numbers.
- $N_{\text{RAN_S}}^k \hat{=} \mathbb{N}^+$, set of all positive natural numbers.
- $N_{\text{Chap_per_Slot}}^k \hat{=} \{3, 4, 6, 8, 9, 12, 24\}$
- $N_{\text{Slot_per_Round}}^k \hat{=} \{6, 8, 9, 12, 16, 18, 24, 32, 36, 48, 72, 96\}$
- $N_{\text{Slot_per_Round}}^k \hat{=} \{6, 8, 9, 12, 16, 18\}$ when the number of responder in each ranging round
- $N_{\text{Responder}}^k \leq 10$.

Table G-1 shows which combinations of $N_{\text{Chap_per_Slot}}^k$ and $N_{\text{Slot_per_Round}}^k$ result in valid ranging session configurations. During the handshake process for the k -th ranging session over the

Bluetooth LE control channel (for details see Section 20.5.2), the Table G-1 is used to negotiate a valid ranging session configuration that maximizes the number of hopping rounds N_{Round}^k for a given set of constraint parameters $N_{\text{Chap_per_Slot}}^k$ and $N_{\text{Slot_per_Round}}^k$. With $N_{\text{Responder}}^k \leq 10$, the only the shaded part of the Table G-1 below will be used during this handshake process.

Table G-1: Number of Valid Ranging Rounds per 96 ms.

Number of Slots (per 96 ms)			$N_{\text{Slot_per_Round}}^k$											
$N_{\text{Chap_per_Slot}}^k$	Resulting T_{Slot}^k		6	8	9	12	16	18	24	32	36	48	72	96
	3	1.000 ms	16	12	N/A	8	6	N/A	4	3	N/A	2	N/A	1
	4	1.333 ms	12	9	8	6	N/A	4	3	N/A	2	N/A	1	N/A
	6	2.000 ms	8	6	N/A	4	3	N/A	2	N/A	N/A	1	N/A	N/A
	8	2.667 ms	6	N/A	4	3	N/A	2	N/A	N/A	1	N/A	N/A	N/A
	9	3.000 ms	N/A	4	N/A	N/A	2	N/A	N/A	1	N/A	N/A	N/A	N/A
	12	4.000 ms	4	3	N/A	2	N/A	N/A	1	N/A	N/A	N/A	N/A	N/A
	24	8.000 ms	2	N/A	N/A	1	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

G.2. Hopping Sequence

This section describes the method to generate the hopping sequence

G.3. Default Hopping Sequence:

This method is mandatory and shall be supported by any DK device.

For the k -th ranging session with N_{Round}^k ranging rounds per block and hopping key

$HOP_Key_RW^k$, the following function is used to generate the hopping round S_k index for ranging block i :

$$S^k(i, HOP_Key_RW^k, N_{\text{Round}}^k) = \left(\left(\left((i + HOP_Key_RW^k) \& 0xFFFF \right)^2 \bmod (2^{16} - 15) \right) N_{\text{Round}}^k \right) \gg 16$$

The initiator shall generate and send the hopping key $HOP_Key_RW^k$ to the responder-device during the ranging session initiation process (see Section 20.5.2).

G.4. AES-Based Hopping Sequence:

This method is optional and may be supported by any DK device. It uses AES to generate the hopping sequence.

1 For the k -th ranging session with N_{Round}^k ranging rounds per block and hopping key
2 $HOP_Key_RW^k$, the following function is used to generate the AES-based hopping round
3 index for ranging block i :

$$4 \quad S^k(i, HOP_Key_RW^k, N_{\text{Round}}^k) = \left(\left(\left(AES(i, HOP_Key_RW^k) \& 0xFFFF \right) N_{\text{Round}}^k \right) \gg 16 \right)$$

5 For the AES function, AES-128 shall be used. Here, the notation AES(plaintext, key) is used
6 where the AES() operation pads each of the inputs to the left with 0x00 to reach the AES block
7 size.

8 **Appendix H. [WCC3]**

9 The following Python code is contributed to IEEE Mentor as in [36] and implements a
10 minimum-phase conversion from [34].

```
11 def min_phase(firseq, fft_pts):
12     """A function to convert a FIR impulse response to min-phase
13
14     A function that calculates the minimum-phase equivalent of a
15     FIR
16     impulse response based on a Discrete Hilbert Transform
17     applied in the frequency domain.
18     """
19     # Recommended fft_pts >= 32768
20     #
21     # For guidelines on the required number of FFT points, see:
22     # N. Damara-Venkata, B. L. Evans and S. R. McCaslin,
23     # "Design of optimal minimum-phase digital FIR filters using
24     # discrete Hilbert transforms," in IEEE Transactions on
25     Signal
26     # Processing, vol. 48, no. 5, pp. 1491-1495, May 2000.
27     max_phase = np.real(np.fft.ifft( \
28         np.exp(hilbert(np.real(np.log(np.fft.fft( \
29             firseq, fft_pts))))))
30     tmp_min_phase = max_phase[::-1] # reverse max phase
31     return tmp_min_phase[0:len(firseq)] # maintain length
32
```

33 **Appendix I. [WCC3]**

34 Pulse shapes using β of 0.45 with 32x oversampling per chip with total span of 8 chips are
35 provided here for informative reference only. The samples are left to right, row by row from

early to late samples. PulseShape 0x0 follows the mathematical equation in Section 21.5 and PulseShape 0x1 is converted from PulseShape 0x0 with the code shown in Appendix H. Implementation may choose to gradually smooth the edge samples toward zero.

1.1. PulseShape 0x0

-0.0089385	-0.0080918	-0.0070627	-0.0058655	-0.0045178	-0.0030411	-0.0014601
0.0001978	0.0019028	0.0036231	0.0053257	0.0069768	0.0085426	0.0099899
0.0112867	0.0124031	0.0133117	0.0139884	0.0144129	0.0145693	0.0144466
0.0140389	0.0133461	0.0123737	0.0111332	0.0096417	0.0079225	0.0060041
0.0039205	0.0017102	-0.0005840	-0.0029158	-0.0052361	-0.0074940	-0.0096374
-0.0116144	-0.0133739	-0.0148670	-0.0160477	-0.0168743	-0.0173102	-0.0173247
-0.0168943	-0.0160030	-0.0146437	-0.0128178	-0.0105365	-0.0078208	-0.0047014
-0.0012189	0.0025766	0.0066255	0.0108598	0.0152038	0.0195746	0.0238837
0.0280380	0.0319412	0.0354955	0.0386030	0.0411676	0.0430968	0.0443030
0.0447061	0.0442347	0.0428277	0.0404366	0.0370264	0.0325774	0.0270859
0.0205660	0.0130496	0.0045875	-0.0047503	-0.0148749	-0.0256783	-0.0370341
-0.0487983	-0.0608102	-0.0728941	-0.0848606	-0.0965085	-0.1076271	-0.1179985
-0.1273998	-0.1356065	-0.1423943	-0.1475430	-0.1508388	-0.1520776	-0.1510679
-0.1476337	-0.1416170	-0.1328811	-0.1213124	-0.1068231	-0.0893529	-0.0688710
-0.0453773	-0.0189032	0.0104872	0.0426970	0.0775967	0.1150248	0.1547883
0.1966642	0.2404012	0.2857218	0.3323244	0.3798865	0.4280672	0.4765109
0.5248505	0.5727115	0.6197154	0.6654839	0.7096429	0.7518264	0.7916805
0.8288673	0.8630684	0.8939885	0.9213591	0.9449406	0.9645258	0.9799418
0.9910521	0.9977580	1.0000000	0.9977580	0.9910521	0.9799418	0.9645258
0.9449406	0.9213591	0.8939885	0.8630684	0.8288673	0.7916805	0.7518264
0.7096429	0.6654839	0.6197154	0.5727115	0.5248505	0.4765109	0.4280672
0.3798865	0.3323244	0.2857218	0.2404012	0.1966642	0.1547883	0.1150248
0.0775967	0.0426970	0.0104872	-0.0189032	-0.0453773	-0.0688710	-0.0893529
-0.1068231	-0.1213124	-0.1328811	-0.1416170	-0.1476337	-0.1510679	-0.1520776
-0.1508388	-0.1475430	-0.1423943	-0.1356065	-0.1273998	-0.1179985	-0.1076271
-0.0965085	-0.0848606	-0.0728941	-0.0608102	-0.0487983	-0.0370341	-0.0256783
-0.0148749	-0.0047503	0.0045875	0.0130496	0.0205660	0.0270859	0.0325774
0.0370264	0.0404366	0.0428277	0.0442347	0.0447061	0.0443030	0.0430968
0.0411676	0.0386030	0.0354955	0.0319412	0.0280380	0.0238837	0.0195746

1	0.0152038	0.0108598	0.0066255	0.0025766	-0.0012189	-0.0047014	-0.0078208
2	-0.0105365	-0.0128178	-0.0146437	-0.0160030	-0.0168943	-0.0173247	-0.0173102
3	-0.0168743	-0.0160477	-0.0148670	-0.0133739	-0.0116144	-0.0096374	-0.0074940
4	-0.0052361	-0.0029158	-0.0005840	0.0017102	0.0039205	0.0060041	0.0079225
5	0.0096417	0.0111332	0.0123737	0.0133461	0.0140389	0.0144466	0.0145693
6	0.0144129	0.0139884	0.0133117	0.0124031	0.0112867	0.0099899	0.0085426
7	0.0069768	0.0053257	0.0036231	0.0019028	0.0001978	-0.0014601	-0.0030411
8	-0.0045178	-0.0058655	-0.0070627	-0.0080918			
9							
10	<i>1.2. PulseShape 0x2</i>						
11	0.0193408	0.0273888	0.0369563	0.0481487	0.0610612	0.0757769	0.0923654
12	0.1108803	0.1313579	0.1538153	0.1782489	0.2046337	0.2329215	0.2630407
13	0.2948957	0.3283664	0.3633086	0.3995544	0.4369126	0.4751700	0.5140929
14	0.5534282	0.5929059	0.6322413	0.6711370	0.7092864	0.7463760	0.7820887
15	0.8161076	0.8481184	0.8778138	0.9048965	0.9290824	0.9501048	0.9677167
16	0.9816948	0.9918417	0.9979891	1.0000000	0.9977710	0.9912337	0.9803566
17	0.9651456	0.9456452	0.9219380	0.8941451	0.8624247	0.8269719	0.7880160
18	0.7458198	0.7006760	0.6529055	0.6028537	0.5508875	0.4973912	0.4427634
19	0.3874123	0.3317516	0.2761968	0.2211602	0.1670473	0.1142522	0.0631537
20	0.0141115	-0.0325377	-0.0764831	-0.1174435	-0.1551694	-0.1894459	-0.2200942
21	-0.2469734	-0.2699813	-0.2890554	-0.3041727	-0.3153495	-0.3226406	-0.3261383
22	-0.3259704	-0.3222984	-0.3153151	-0.3052419	-0.2923256	-0.2768351	-0.2590582
23	-0.2392974	-0.2178666	-0.1950870	-0.1712832	-0.1467796	-0.1218960	-0.0969449
24	-0.0722271	-0.0480290	-0.0246194	-0.0022470	0.0188624	0.0385076	0.0565141
25	0.0727348	0.0870515	0.0993751	0.1096460	0.1178335	0.1239357	0.1279780
26	0.1300121	0.1301142	0.1283833	0.1249388	0.1199185	0.1134756	0.1057763
27	0.0969971	0.0873217	0.0769383	0.0660371	0.0548067	0.0434326	0.0320936
28	0.0209603	0.0101925	-0.0000627	-0.0096722	-0.0185187	-0.0265018	-0.0335384
29	-0.0395639	-0.0445319	-0.0484147	-0.0512024	-0.0529029	-0.0535408	-0.0531563
30	-0.0518044	-0.0495529	-0.0464813	-0.0426788	-0.0382426	-0.0332763	-0.0278875
31	-0.0221866	-0.0162841	-0.0102898	-0.0043100	0.0015533	0.0072041	0.0125540
32	0.0175230	0.0220408	0.0260473	0.0294936	0.0323420	0.0345664	0.0361523

1	0.0370968	0.0374078	0.0371038	0.0362131	0.0347730	0.0328289	0.0304330
2	0.0276434	0.0245228	0.0211373	0.0175550	0.0138452	0.0100764	0.0063158
3	0.0026280	-0.0009263	-0.0042910	-0.0074151	-0.0102543	-0.0127706	-0.0149333
4	-0.0167193	-0.0181129	-0.0191063	-0.0196988	-0.0198975	-0.0197161	-0.0191746
5	-0.0182993	-0.0171212	-0.0156759	-0.0140028	-0.0121439	-0.0101431	-0.0080455
6	-0.0058961	-0.0037395	-0.0016186	0.0004259	0.0023562	0.0041382	0.0057420
7	0.0071423	0.0083185	0.0092556	0.0099438	0.0103786	0.0105611	0.0104975
8	0.0101990	0.0096813	0.0089643	0.0080718	0.0070304	0.0058694	0.0046200
9	0.0033145	0.0019857	0.0006661	-0.0006126	-0.0018203	-0.0029293	-0.0039144
10	-0.0047540	-0.0054301	-0.0059290	-0.0062416	-0.0063635	-0.0062950	-0.0060415
11	-0.0056132	-0.0050251	-0.0042966	-0.0034510	-0.0025154	-0.0015200	-0.0004973
12	0.00051848	0.00149222	0.00238888	0.00317447	0.00381706	0.00428794	0.00456281
13	0.00462307	0.00445708	0.00406159	0.00344317	0.00261972	0.00162203	0.00049542
14	-0.0006985	-0.0018802	-0.0029501	-0.0037867	-0.0042448	-0.0041529	-0.0033108
15	-0.0014877	0.00158115	0.00619674	0.0000000319			
16							

Appendix J. [WCC3]

J.1. UWB Test Vector

Values are Hexadecimal unless labeled Decimal.

STSIndex0_Decimal: 123456789

STSIndex0_Hex_BigEndian: 075bcd15

URSK: ed07a80d2beb00f785af2627c96ae7c118504243cb2c3226b3679daa0f7e616c

mUPSK1: 94ce7a78b643d0cac4e2e4f8b5f55a4d

mUPSK2: a93a052ba546f74aa1b771dfb8c89f503a199bcc7af64ad2881c94b320a1243e

UAD: 8b779d0709d9af2c324320afb5301a8f

Keysource-low: 8b77

Keysource-high: 9d07

DestinationShortAddress: 09d9

SourceLongAddress: af2c324320afb530

mURSK: 1cac2a6a5b04490c86089bb989c7d35987c47efc3fff71133bd98e6d7966de70

Salt: c56ee5c153daf28f917c6081c8bb63ca

PaddedSalt: 00000000000000000000000000000000c56ee5c153daf28f917c6081c8bb63ca

Selected_Protocol_Version: 0100

Selected_UWB_Config_Id: 0000

```
1  UWB_Session_Id: 12345678
2  STS_Index0: 075bcd15
3  Number_Responder_Nodes: 01
4  Session_RAN_Multiplier: 0a
5  Number_Slot_per_Round: 06
6  Number_Chaps_per_Slot: 06
7  Selected_PulseShape_Combo: 00
8  Ranging Configuration: 0100000012345678075bcd15010a060600
9  SaltedHash: 79e865186cbd864b955682c51cdd34f8
10 STSIndex_Decimal: 123456789
11 KDF Cascade Context:
12 0000000000010101000100010100010101010000010100010000000100010001
13 URSK_KT: d8c951ca6029a8ff5c89199b6ae07387301056ac3462f93db1afb2009cd89d2f
14 dURSK: 4578aa54d663cb5aa88070cd01c604a7
15 dUDSK: 932e0f6d9bb393096fe24cc79a704447
16
17 Second Block (Block index i=1), first Ranging Round (Round index s=0)
18 STSIndex_Decimal: 123457269
19 KDF Cascade Context:
20 0000000000010101000100010100010101010000010101000101010100010001
21 URSK_KT: ebd253fbf2c57294284de578fee5de984f3489d740e6fd5ee50af537c714af6e
22 dURSK: 9d40e2080db7d2cda114838189d7ae9f
23 dUDSK: 011adc74ba889e7e59a8ee61889d3fec
```

J.2. UWB Test Vector without hopping

Based on the UWB Test Vector of J.1 the following exchange of UWB PPDU's may occur:

```
26 Values are Hexadecimal.
27 First Ranging block:
28   STS_Index for Pre Poll: 07 5B CD 15
29   Frame Counter for Pre Poll: 00 00 00 00
30   CMAC-Data(URSK_KT): 00 00 00 01 55 52 53 4B 5F 4B 54 00 00 00 00 00
31                       00 01 01 01 00 01 00 01 01 00 01 01 01 01 00 00
32                       01 01 00 01 00 00 00 01 00 01 00 01 00 00 01 00
33   CMAC-Data(URSK_KT): 00 00 00 02 55 52 53 4B 5F 4B 54 00 00 00 00 00
34                       00 01 01 01 00 01 00 01 01 00 01 01 01 01 00 00
35                       01 01 00 01 00 00 00 01 00 01 00 01 00 00 01 00
36   URSK_KT: D8 C9 51 CA 60 29 A8 FF 5C 89 19 9B 6A E0 73 87
37           30 10 56 AC 34 62 F9 3D B1 AF B2 00 9C D8 9D 2F
38   CMAC-Data(URSK): 00 00 00 01 55 52 53 4B 00 79 E8 65 18 6C BD 86
```



```
1          4B 95 56 82 C5 1C DD 34 F8 00 00 00 00 00 80
2    dURSK: 45 78 AA 54 D6 63 CB 5A A8 80 70 CD 01 C6 04 A7
3          CMAC-Data(UDSK): 00 00 00 01 55 44 53 4B 00 79 E8 65 18 6C BD 86
4          4B 95 56 82 C5 1C DD 34 F8 00 00 00 00 00 80
5    dUDSK: 93 2E 0F 6D 9B B3 93 09 6F E2 4C C7 9A 70 44 47
6    PrePoll MHR: 49 2B D9 09 16 00 00 00 00 77 8B 07 9D AA 05 00
7          69 DF 04 01 0D 80 3F
8    PrePoll Payload: 78 56 34 12 16 CD 5B 07 00 00 00 00 00
9          AES-CCM* key K: 94 CE 7A 78 B6 43 D0 CA C4 E2 E4 F8 B5 F5 5A 4D
10         Nonce N: AF 2C 32 43 20 AF B5 30 00 00 00 00 06
11         AddAuthData: 00 17 49 2B D9 09 16 00 00 00 00 77 8B 07 9D AA
12                 05 00 69 DF 04 01 0D 80 3F 00 00 00 00 00 00
13         PlainTextData: 78 56 34 12 16 CD 5B 07 00 00 00 00 00 00 00
14         AuthData: 00 17 49 2B D9 09 16 00 00 00 00 77 8B 07 9D AA
15                 05 00 69 DF 04 01 0D 80 3F 00 00 00 00 00 00
16                 78 56 34 12 16 CD 5B 07 00 00 00 00 00 00 00
17         B0: 59 AF 2C 32 43 20 AF B5 30 00 00 00 00 06 00 0D
18         X: DD 38 D8 1E 45 5A 1B AD A3 58 8C 62 6C 90 EF E7
19                 A2 AB 2B AE DE C2 E5 B4 D1 42 9B AF 4D 48 2F 9B
20                 E5 2E A2 52 9C 05 9F DD 0F 78 B9 BB A5 54 4D B8
21                 CB 2A A9 D5 17 55 E2 40 00 93 E1 78 86 5F 63 D2
22         T: CB 2A A9 D5 17 55 E2 40
23         A0: 01 AF 2C 32 43 20 AF B5 30 00 00 00 00 06 00 00
24         A1: 01 AF 2C 32 43 20 AF B5 30 00 00 00 00 06 00 01
25         C: 4A 87 F6 F8 FD EC EF EA EA 19 34 00 43 4D CC 06
26         S0: 11 43 A9 A2 0F F6 E6 C3 B6 CE 61 6C 47 85 65 96
27         CipherText: 32 D1 C2 EA EB 21 B4 ED EA 19 34 00 43
28         U: DA 69 00 77 18 A3 04 83
29     PrePoll encrypted Payload: 32 D1 C2 EA EB 21 B4 ED EA 19 34 00 43
30                 DA 69 00 77 18 A3 04 83
31     STS-V for      POLL PPDU: 79 E8 65 18 6C BD 86 4B 9C B2 4F DB 1C DD 34 F8 -
32                 1C DD 35 17
33     STS for      POLL PPDU: E8 AE 41 D3 4E 61 87 10 95 A6 2B 65 EF 1F 15 50 ..
34                 9D 5F 97 B1 EB 2B 09 55 40 B2 87 72 4B D6 88 76
35     STS-V for 1. RESPONSE PPDU: 79 E8 65 18 6C BD 86 4B 9C B2 4F DC 1C DD 34 F8 -
36                 1C DD 35 17
37     STS for 1. RESPONSE PPDU: FA 26 6A E7 77 40 FA 4A E6 81 2A F3 E9 69 9D A2 ..
38                 1D 71 EA 7E 7D 0E A1 47 CC EB F3 18 DF 1D EC 9A
39     STS-V for      FINAL PPDU: 79 E8 65 18 6C BD 86 4B 9C B2 4F DD 1C DD 34 F8 -
```

```
1          1C DD 35 17
2      STS for          FINAL PPDU: 51 76 79 08 78 9D 7D B7 49 1C 87 4B 8E 2E AF 7E ..
3          87 8E C1 D3 3E D9 79 80 E5 E3 2D 7E 17 97 46 E0
4      FinaData MHR: 49 2B D9 09 16 01 00 00 00 77 8B 07 9D AA 05 00
5          69 DF 04 02 19 80 3F
6      FinaData Payload: 78 56 34 12 00 00 00 00 18 CD 5B 07 00 00 3C
7          0F 01 00 FF 04 9E 07 29 00
8      AES-CCM* key K: 93 2E 0F 6D 9B B3 93 09 6F E2 4C C7 9A 70 44 47
9      Nonce N: AF 2C 32 43 20 AF B5 30 00 00 00 01 06
10     AddAuthData: 00 17 49 2B D9 09 16 01 00 00 00 77 8B 07 9D AA
11          05 00 69 DF 04 02 19 80 3F 00 00 00 00 00 00 00
12     PlainTextData: 78 56 34 12 00 00 00 00 18 CD 5B 07 00 00 3C
13          0F 01 00 FF 04 9E 07 29 00 00 00 00 00 00 00 00
14     AuthData: 00 17 49 2B D9 09 16 01 00 00 00 77 8B 07 9D AA
15          05 00 69 DF 04 02 19 80 3F 00 00 00 00 00 00 00
16          78 56 34 12 00 00 00 00 18 CD 5B 07 00 00 3C
17          0F 01 00 FF 04 9E 07 29 00 00 00 00 00 00 00 00
18     B0: 59 AF 2C 32 43 20 AF B5 30 00 00 00 01 06 00 19
19     X: 1A C5 15 F3 E1 77 DB A3 1D 20 DC 57 35 10 69 AD
20          3A 51 46 95 AD A3 CB 01 EC 16 C4 BB E2 3F 28 D5
21          BA CC 1B 33 47 D5 13 BB 39 40 56 13 78 AC C9 77
22          8B EF 16 7F BC DA 85 BD AF B6 BB 8B 80 F0 4C E3
23          B0 DA 6F BE 11 35 0C 2F 5C 80 50 DF 0B 7C CF D0
24     T: B0 DA 6F BE 11 35 0C 2F
25     A0: 01 AF 2C 32 43 20 AF B5 30 00 00 00 01 06 00 00
26     A1: 01 AF 2C 32 43 20 AF B5 30 00 00 00 01 06 00 01
27     A2: 01 AF 2C 32 43 20 AF B5 30 00 00 00 01 06 00 02
28     C: E2 55 9D 5B EA 8C 3E 87 44 4C 08 F3 7B 2D 5F 10
29          3C F2 DB 4A AB 57 E7 43 B7 78 D5 4D C2 AD C6 38
30     S0: 2B 36 52 7B F4 57 8F FD CB C0 82 81 DD CD BB 6F
31     CipherText: 9A 03 A9 49 EA 8C 3E 87 44 54 C5 A8 7C 2D 5F 2C
32          33 F3 DB B5 AF C9 E0 6A B7
33     U: 9B EC 3D C5 E5 62 83 D2
34     FinaData encrypted Payload: 9A 03 A9 49 EA 8C 3E 87 44 54 C5 A8 7C 2D 5F 2C
35          33 F3 DB B5 AF C9 E0 6A B7 9B EC 3D C5 E5 62 83 D2
36     Second Ranging block:
37     STS_Index for Pre Poll: 07 5B CE F5
38     Frame Counter for Pre Poll: 00 00 00 02
39     CMAC-Data(URSK_KT): 00 00 00 01 55 52 53 4B 5F 4B 54 00 00 00 00 00
```

```
1          00 01 01 01 00 01 00 01 01 00 01 01 01 01 00 00
2          01 01 01 00 01 01 01 01 00 01 00 01 00 00 01 00
3      CMAC-Data(URSK_KT): 00 00 00 02 55 52 53 4B 5F 4B 54 00 00 00 00 00
4          00 01 01 01 00 01 00 01 01 00 01 01 01 01 00 00
5          01 01 01 00 01 01 01 01 00 01 00 01 00 00 01 00
6      URSK_KT: EB D2 53 FB F2 C5 72 94 28 4D E5 78 FE E5 DE 98
7          4F 34 89 D7 40 E6 FD 5E E5 0A F5 37 C7 14 AF 6E
8      CMAC-Data(URSK): 00 00 00 01 55 52 53 4B 00 79 E8 65 18 6C BD 86
9          4B 95 56 82 C5 1C DD 34 F8 00 00 00 00 00 00 80
10     dURSK: 9D 40 E2 08 0D B7 D2 CD A1 14 83 81 89 D7 AE 9F
11     CMAC-Data(UDSK): 00 00 00 01 55 44 53 4B 00 79 E8 65 18 6C BD 86
12          4B 95 56 82 C5 1C DD 34 F8 00 00 00 00 00 00 80
13     dUDSK: 01 1A DC 74 BA 88 9E 7E 59 A8 EE 61 88 9D 3F EC
14     PrePoll MHR: 49 2B D9 09 16 02 00 00 00 77 8B 07 9D AA 05 00
15          69 DF 04 01 0D 80 3F
16     PrePoll Payload: 78 56 34 12 F6 CE 5B 07 01 00 00 00 00
17     AES-CCM* key K: 94 CE 7A 78 B6 43 D0 CA C4 E2 E4 F8 B5 F5 5A 4D
18     Nonce N: AF 2C 32 43 20 AF B5 30 00 00 00 02 06
19     AddAuthData: 00 17 49 2B D9 09 16 02 00 00 00 77 8B 07 9D AA
20          05 00 69 DF 04 01 0D 80 3F 00 00 00 00 00 00 00
21     PlainTextData: 78 56 34 12 F6 CE 5B 07 01 00 00 00 00 00 00 00
22     AuthData: 00 17 49 2B D9 09 16 02 00 00 00 77 8B 07 9D AA
23          05 00 69 DF 04 01 0D 80 3F 00 00 00 00 00 00 00
24          78 56 34 12 F6 CE 5B 07 01 00 00 00 00 00 00 00
25     B0: 59 AF 2C 32 43 20 AF B5 30 00 00 00 02 06 00 0D
26     X: 2F A9 D9 44 AA 74 A0 6F C7 CD AD 33 25 74 8A 5D
27          7E 8C 86 E7 4D 2D EF 05 8F AC AE 3C 02 88 3C 85
28          BC 7F 2D BB EE FE 33 9B 1B 77 F0 85 CE B1 0A AD
29          94 EF E7 D4 A6 04 BE 54 1F 47 96 EE E7 23 BA 94
30     T: 94 EF E7 D4 A6 04 BE 54
31     A0: 01 AF 2C 32 43 20 AF B5 30 00 00 00 02 06 00 00
32     A1: 01 AF 2C 32 43 20 AF B5 30 00 00 00 02 06 00 01
33     C: BA 38 C2 2E 8C E7 D5 C8 DC 89 27 31 E7 77 D0 84
34     S0: 6B 6D EE 0F 3E D9 56 7B C1 69 45 5B 33 14 8F 4D
35     CipherText: C2 6E F6 3C 7A 29 8E CF DD 89 27 31 E7
36     U: FF 82 09 DB 98 DD E8 2F
37     PrePoll encrypted Payload: C2 6E F6 3C 7A 29 8E CF DD 89 27 31 E7
38          FF 82 09 DB 98 DD E8 2F
39     STS-V for          POLL PPDU: 79 E8 65 18 6C BD 86 4B 9C B2 51 BB 1C DD 34 F8 -
```

```
1          1C DD 35 17
2      STS for      POLL PPDU: BE 4E 6F 9D 5B 1C 4B 76 51 AF 10 47 40 BA 22 C3 ..
3          45 24 6A 04 FB AE 29 81 58 D2 70 E9 1D EB F6 C3
4      STS-V for 1. RESPONSE PPDU: 79 E8 65 18 6C BD 86 4B 9C B2 51 BC 1C DD 34 F8 -
5          1C DD 35 17
6      STS for 1. RESPONSE PPDU: 68 D3 38 09 A2 6A 26 08 47 0E D7 CC 14 67 2F D1 ..
7          59 F4 F9 EE 2D EC DB CE 03 0D 76 4C 75 A1 7C FA
8      STS-V for      FINAL PPDU: 79 E8 65 18 6C BD 86 4B 9C B2 51 BD 1C DD 34 F8 -
9          1C DD 35 17
10     STS for      FINAL PPDU: C0 B1 17 8D 50 60 C2 D6 70 0C C2 18 29 F6 BC 45 ..
11          B7 26 0E C9 6E 2C DE BB 1E 2E 2D 42 D3 58 26 39
12     FinaData MHR: 49 2B D9 09 16 03 00 00 00 77 8B 07 9D AA 05 00
13          69 DF 04 02 19 80 3F
14     FinaData Payload: 78 56 34 12 01 00 00 00 00 F8 CE 5B 07 00 00 3C
15          0F 01 00 04 05 9E 07 29 00
16     AES-CCM* key K: 01 1A DC 74 BA 88 9E 7E 59 A8 EE 61 88 9D 3F EC
17     Nonce N: AF 2C 32 43 20 AF B5 30 00 00 00 03 06
18     AddAuthData: 00 17 49 2B D9 09 16 03 00 00 00 77 8B 07 9D AA
19          05 00 69 DF 04 02 19 80 3F 00 00 00 00 00 00 00
20     PlainTextData: 78 56 34 12 01 00 00 00 00 F8 CE 5B 07 00 00 3C
21          0F 01 00 04 05 9E 07 29 00 00 00 00 00 00 00 00
22     AuthData: 00 17 49 2B D9 09 16 03 00 00 00 77 8B 07 9D AA
23          05 00 69 DF 04 02 19 80 3F 00 00 00 00 00 00 00
24          78 56 34 12 01 00 00 00 00 F8 CE 5B 07 00 00 3C
25          0F 01 00 04 05 9E 07 29 00 00 00 00 00 00 00 00
26     B0: 59 AF 2C 32 43 20 AF B5 30 00 00 00 03 06 00 19
27     X: AA 21 1A C9 71 A4 FA B8 39 20 22 11 DA 71 B8 B1
28          44 44 62 2D E7 20 05 78 AF F1 86 A8 30 6C 5C B8
29          A8 00 46 D8 0A 92 EB F0 3D CD F5 AD F3 E3 92 F2
30          B0 77 A4 86 F9 E3 0D AE EB AC C8 67 BF 6A 45 32
31          42 DF 7B 1E B6 60 4B 9A 95 DC D0 49 82 33 93 B3
32     T: 42 DF 7B 1E B6 60 4B 9A
33     A0: 01 AF 2C 32 43 20 AF B5 30 00 00 00 03 06 00 00
34     A1: 01 AF 2C 32 43 20 AF B5 30 00 00 00 03 06 00 01
35     A2: 01 AF 2C 32 43 20 AF B5 30 00 00 00 03 06 00 02
36     C: 89 E7 B9 4F 63 EA 41 AC BA 5A BF 44 41 71 6C 01
37          0C C4 F9 0B CC 09 FB 62 66 80 74 6F 8E 5A 76 F0
38     S0: 09 7D 64 94 03 79 2F 64 E3 48 D8 DC BA 59 AC A5
39     CipherText: F1 B1 8D 5D 62 EA 41 AC BA A2 71 1F 46 71 6C 3D
```

```
03 C5 F9 0F C9 97 FC 4B 66
U: 4B A2 1F 8A B5 19 64 FE
FinaData encrypted Payload: F1 B1 8D 5D 62 EA 41 AC BA A2 71 1F 46 71 6C 3D
03 C5 F9 0F C9 97 FC 4B 66 4B A2 1F 8A B5 19 64 FE
```

J.3. UWB Test Vector with continuous hopping

Based on the UWB Test Vector of J.1 the following exchange of UWB PPDU's may occur:

```
Values are Hexadecimal.
Hop_Mode_Key for Default Hopping Sequence: CC 5D D7 9F
Rounds per Block (Nround): 00 50
First Ranging block:
STS_Index for Pre Poll: 07 5B CD 15
Frame Counter for Pre Poll: 00 00 00 00
CMAC-Data(URSK_KT): 00 00 00 01 55 52 53 4B 5F 4B 54 00 00 00 00 00
00 01 01 01 00 01 00 01 01 00 01 01 01 01 00 00
01 01 00 01 00 00 00 01 00 01 00 01 00 00 01 00
CMAC-Data(URSK_KT): 00 00 00 02 55 52 53 4B 5F 4B 54 00 00 00 00 00
00 01 01 01 00 01 00 01 01 00 01 01 01 01 00 00
01 01 00 01 00 00 00 01 00 01 00 01 00 00 01 00
URSK_KT: D8 C9 51 CA 60 29 A8 FF 5C 89 19 9B 6A E0 73 87
30 10 56 AC 34 62 F9 3D B1 AF B2 00 9C D8 9D 2F
CMAC-Data(URSK): 00 00 00 01 55 52 53 4B 00 79 E8 65 18 6C BD 86
4B 95 56 82 C5 1C DD 34 F8 00 00 00 00 00 00 80
dURSK: 45 78 AA 54 D6 63 CB 5A A8 80 70 CD 01 C6 04 A7
CMAC-Data(UDSK): 00 00 00 01 55 44 53 4B 00 79 E8 65 18 6C BD 86
4B 95 56 82 C5 1C DD 34 F8 00 00 00 00 00 00 80
dUDSK: 93 2E 0F 6D 9B B3 93 09 6F E2 4C C7 9A 70 44 47
PrePoll MHR: 49 2B D9 09 16 00 00 00 00 77 8B 07 9D AA 05 00
69 DF 04 01 0D 80 3F
PrePoll Payload: 78 56 34 12 16 CD 5B 07 00 00 00 00 00 00
AES-CCM* key K: 94 CE 7A 78 B6 43 D0 CA C4 E2 E4 F8 B5 F5 5A 4D
Nonce N: AF 2C 32 43 20 AF B5 30 00 00 00 00 06
AddAuthData: 00 17 49 2B D9 09 16 00 00 00 00 77 8B 07 9D AA
05 00 69 DF 04 01 0D 80 3F 00 00 00 00 00 00 00
PlainTextData: 78 56 34 12 16 CD 5B 07 00 00 00 00 00 00 00 00
AuthData: 00 17 49 2B D9 09 16 00 00 00 00 77 8B 07 9D AA
05 00 69 DF 04 01 0D 80 3F 00 00 00 00 00 00 00
78 56 34 12 16 CD 5B 07 00 00 00 00 00 00 00 00
B0: 59 AF 2C 32 43 20 AF B5 30 00 00 00 00 06 00 0D
```

```
1      X: DD 38 D8 1E 45 5A 1B AD A3 58 8C 62 6C 90 EF E7
2      A2 AB 2B AE DE C2 E5 B4 D1 42 9B AF 4D 48 2F 9B
3      E5 2E A2 52 9C 05 9F DD 0F 78 B9 BB A5 54 4D B8
4      CB 2A A9 D5 17 55 E2 40 00 93 E1 78 86 5F 63 D2
5      T: CB 2A A9 D5 17 55 E2 40
6      A0: 01 AF 2C 32 43 20 AF B5 30 00 00 00 00 06 00 00
7      A1: 01 AF 2C 32 43 20 AF B5 30 00 00 00 00 06 00 01
8      C: 4A 87 F6 F8 FD EC EF EA EA 19 34 00 43 4D CC 06
9      S0: 11 43 A9 A2 0F F6 E6 C3 B6 CE 61 6C 47 85 65 96
10     CipherText: 32 D1 C2 EA EB 21 B4 ED EA 19 34 00 43
11     U: DA 69 00 77 18 A3 04 83
12     PrePoll encrypted Payload: 32 D1 C2 EA EB 21 B4 ED EA 19 34 00 43
13     DA 69 00 77 18 A3 04 83
14     STS-V for      POLL PPDU: 79 E8 65 18 6C BD 86 4B 9C B2 4F DB 1C DD 34 F8 -
15     1C DD 35 17
16     STS for      POLL PPDU: E8 AE 41 D3 4E 61 87 10 95 A6 2B 65 EF 1F 15 50 ..
17     9D 5F 97 B1 EB 2B 09 55 40 B2 87 72 4B D6 88 76
18     STS-V for 1. RESPONSE PPDU: 79 E8 65 18 6C BD 86 4B 9C B2 4F DC 1C DD 34 F8 -
19     1C DD 35 17
20     STS for 1. RESPONSE PPDU: FA 26 6A E7 77 40 FA 4A E6 81 2A F3 E9 69 9D A2 ..
21     1D 71 EA 7E 7D 0E A1 47 CC EB F3 18 DF 1D EC 9A
22     STS-V for      FINAL PPDU: 79 E8 65 18 6C BD 86 4B 9C B2 4F DD 1C DD 34 F8 -
23     1C DD 35 17
24     STS for      FINAL PPDU: 51 76 79 08 78 9D 7D B7 49 1C 87 4B 8E 2E AF 7E ..
25     87 8E C1 D3 3E D9 79 80 E5 E3 2D 7E 17 97 46 E0
26     Round Index of next Ranging Block (Default Hopping): 00 3E
27     FinaData MHR: 49 2B D9 09 16 01 00 00 00 77 8B 07 9D AA 05 00
28     69 DF 04 02 19 80 3F
29     FinaData Payload: 78 56 34 12 00 00 01 3E 00 18 CD 5B 07 00 00 3C
30     0F 01 00 C1 04 9E 07 2C 00
31     AES-CCM* key K: 93 2E 0F 6D 9B B3 93 09 6F E2 4C C7 9A 70 44 47
32     Nonce N: AF 2C 32 43 20 AF B5 30 00 00 00 01 06
33     AddAuthData: 00 17 49 2B D9 09 16 01 00 00 00 77 8B 07 9D AA
34     05 00 69 DF 04 02 19 80 3F 00 00 00 00 00 00 00
35     PlainTextData: 78 56 34 12 00 00 01 3E 00 18 CD 5B 07 00 00 3C
36     0F 01 00 C1 04 9E 07 2C 00 00 00 00 00 00 00 00
37     AuthData: 00 17 49 2B D9 09 16 01 00 00 00 77 8B 07 9D AA
38     05 00 69 DF 04 02 19 80 3F 00 00 00 00 00 00 00
39     78 56 34 12 00 00 01 3E 00 18 CD 5B 07 00 00 3C
```

```

      0F 01 00 C1 04 9E 07 2C 00 00 00 00 00 00 00 00
B0: 59 AF 2C 32 43 20 AF B5 30 00 00 00 01 06 00 19
X: 1A C5 15 F3 E1 77 DB A3 1D 20 DC 57 35 10 69 AD
      3A 51 46 95 AD A3 CB 01 EC 16 C4 BB E2 3F 28 D5
      BA CC 1B 33 47 D5 13 BB 39 40 56 13 78 AC C9 77
      F0 6B A2 25 F6 49 B4 7C EF C1 6B F9 0C C7 82 04
      64 97 62 53 2E CE 36 69 95 42 5D A6 CE 40 95 54
T: 64 97 62 53 2E CE 36 69
A0: 01 AF 2C 32 43 20 AF B5 30 00 00 00 01 06 00 00
A1: 01 AF 2C 32 43 20 AF B5 30 00 00 00 01 06 00 01
A2: 01 AF 2C 32 43 20 AF B5 30 00 00 00 01 06 00 02
C: E2 55 9D 5B EA 8C 3E 87 44 4C 08 F3 7B 2D 5F 10
      3C F2 DB 4A AB 57 E7 43 B7 78 D5 4D C2 AD C6 38
S0: 2B 36 52 7B F4 57 8F FD CB C0 82 81 DD CD BB 6F
CipherText: 9A 03 A9 49 EA 8C 3F B9 44 54 C5 A8 7C 2D 5F 2C
      33 F3 DB 8B AF C9 E0 6F B7
U: 4F A1 30 28 DA 99 B9 94
FinaData encrypted Payload: 9A 03 A9 49 EA 8C 3F B9 44 54 C5 A8 7C 2D 5F 2C
      33 F3 DB 8B AF C9 E0 6F B7 4F A1 30 28 DA 99 B9 94
```

Second Ranging block:

```

STS_Index for Pre Poll: 07 5B D0 69
Frame Counter for Pre Poll: 00 00 00 02
      CMAC-Data(URSK_KT): 00 00 00 01 55 52 53 4B 5F 4B 54 00 00 00 00 00
      00 01 01 01 00 01 00 01 01 00 01 01 01 01 00 01
      00 00 00 00 00 01 01 00 01 00 00 01 00 00 01 00
      CMAC-Data(URSK_KT): 00 00 00 02 55 52 53 4B 5F 4B 54 00 00 00 00 00
      00 01 01 01 00 01 00 01 01 00 01 01 01 01 00 01
      00 00 00 00 00 01 01 00 01 00 00 01 00 00 01 00
URSK_KT: 06 32 7C C5 A3 6C AF 38 6A E3 F1 3F C8 CC A4 40
      42 7A A6 D9 A4 71 A6 E8 9C E4 63 22 F5 79 FE 73
      CMAC-Data(URSK): 00 00 00 01 55 52 53 4B 00 79 E8 65 18 6C BD 86
      4B 95 56 82 C5 1C DD 34 F8 00 00 00 00 00 00 80
dURSK: 74 36 F1 90 14 51 4D C9 6F 90 01 FF 55 EC 87 47
      CMAC-Data(UDSK): 00 00 00 01 55 44 53 4B 00 79 E8 65 18 6C BD 86
      4B 95 56 82 C5 1C DD 34 F8 00 00 00 00 00 00 80
dUDSK: 42 C8 4F 42 03 CE 36 DF 09 58 FF 21 CB 39 F2 F0
PrePoll MHR: 49 2B D9 09 16 02 00 00 00 77 8B 07 9D AA 05 00
      69 DF 04 01 0D 80 3F
PrePoll Payload: 78 56 34 12 6A D0 5B 07 01 00 01 3E 00
```

```
1      AES-CCM* key K: 94 CE 7A 78 B6 43 D0 CA C4 E2 E4 F8 B5 F5 5A 4D
2      Nonce N: AF 2C 32 43 20 AF B5 30 00 00 02 06
3      AddAuthData: 00 17 49 2B D9 09 16 02 00 00 00 77 8B 07 9D AA
4                  05 00 69 DF 04 01 0D 80 3F 00 00 00 00 00 00
5      PlainTextData: 78 56 34 12 6A D0 5B 07 01 00 01 3E 00 00 00 00
6      AuthData: 00 17 49 2B D9 09 16 02 00 00 00 77 8B 07 9D AA
7                  05 00 69 DF 04 01 0D 80 3F 00 00 00 00 00 00
8                  78 56 34 12 6A D0 5B 07 01 00 01 3E 00 00 00 00
9      B0: 59 AF 2C 32 43 20 AF B5 30 00 00 02 06 00 0D
10     X: 2F A9 D9 44 AA 74 A0 6F C7 CD AD 33 25 74 8A 5D
11         7E 8C 86 E7 4D 2D EF 05 8F AC AE 3C 02 88 3C 85
12         BC 7F 2D BB EE FE 33 9B 1B 77 F0 85 CE B1 0A AD
13         4D 10 94 4C 36 D7 C3 B4 5A 3F D1 A6 51 8A B4 30
14     T: 4D 10 94 4C 36 D7 C3 B4
15     A0: 01 AF 2C 32 43 20 AF B5 30 00 00 02 06 00 00
16     A1: 01 AF 2C 32 43 20 AF B5 30 00 00 02 06 00 01
17     C: BA 38 C2 2E 8C E7 D5 C8 DC 89 27 31 E7 77 D0 84
18     S0: 6B 6D EE 0F 3E D9 56 7B C1 69 45 5B 33 14 8F 4D
19     CipherText: C2 6E F6 3C E6 37 8E CF DD 89 26 0F E7
20     U: 26 7D 7A 43 08 0E 95 CF
21     PrePoll encrypted Payload: C2 6E F6 3C E6 37 8E CF DD 89 26 0F E7
22                                     26 7D 7A 43 08 0E 95 CF
23     STS-V for      POLL PPDU: 79 E8 65 18 6C BD 86 4B 9C B2 53 2F 1C DD 34 F8 -
24                                     1C DD 35 17
25     STS for      POLL PPDU: 8F 33 3E 3D D2 5C 0F 57 EB 8B 25 9D 3D 73 DB BC ..
26                                     F4 12 FA A3 01 20 83 02 FF E0 04 AC 43 60 D3 C5
27     STS-V for 1. RESPONSE PPDU: 79 E8 65 18 6C BD 86 4B 9C B2 53 30 1C DD 34 F8 -
28                                     1C DD 35 17
29     STS for 1. RESPONSE PPDU: C7 2B 1A 1B CE 0C 86 C8 E8 28 2A 0B 44 3F F5 FB ..
30                                     B3 CE 69 49 6D FD 40 61 D8 DA B9 DA 6A FA 88 D5
31     STS-V for      FINAL PPDU: 79 E8 65 18 6C BD 86 4B 9C B2 53 31 1C DD 34 F8 -
32                                     1C DD 35 17
33     STS for      FINAL PPDU: 41 B5 25 D1 2F DE 4A DC 74 02 F5 EB F8 15 28 AB ..
34                                     B2 58 5D 96 82 ED 93 A8 E3 2C FE A4 37 EA 00 D5
35     Round Index of next Ranging Block (Default Hopping): 00 25
36     FinaData MHR: 49 2B D9 09 16 03 00 00 00 77 8B 07 9D AA 05 00
37                 69 DF 04 02 19 80 3F
38     FinaData Payload: 78 56 34 12 01 00 01 25 00 6C D0 5B 07 00 00 3C
39                 0F 01 00 B1 04 9E 07 2C 00
```



```
AES-CCM* key K: 42 C8 4F 42 03 CE 36 DF 09 58 FF 21 CB 39 F2 F0
Nonce N: AF 2C 32 43 20 AF B5 30 00 00 00 03 06
AddAuthData: 00 17 49 2B D9 09 16 03 00 00 00 77 8B 07 9D AA
              05 00 69 DF 04 02 19 80 3F 00 00 00 00 00 00
PlainTextData: 78 56 34 12 01 00 01 25 00 6C D0 5B 07 00 00 3C
              0F 01 00 B1 04 9E 07 2C 00 00 00 00 00 00 00
AuthData: 00 17 49 2B D9 09 16 03 00 00 00 77 8B 07 9D AA
           05 00 69 DF 04 02 19 80 3F 00 00 00 00 00 00
           78 56 34 12 01 00 01 25 00 6C D0 5B 07 00 00 3C
           0F 01 00 B1 04 9E 07 2C 00 00 00 00 00 00 00
B0: 59 AF 2C 32 43 20 AF B5 30 00 00 00 03 06 00 19
X: F1 F4 F4 BB 98 C3 45 B1 A1 39 09 05 50 0C 89 B2
   BB 2C B1 43 80 48 6B E5 2B BD E6 E4 70 72 ED EA
   62 26 07 E2 92 0C B7 C0 C8 5A 3C 75 04 5F 2C D7
   C8 67 6D FA 41 A6 3A 92 B5 8B 9D 63 7F B1 9E C8
   4B BD B3 37 B3 B0 89 34 E2 15 B7 48 14 42 64 65
T: 4B BD B3 37 B3 B0 89 34
A0: 01 AF 2C 32 43 20 AF B5 30 00 00 00 03 06 00 00
A1: 01 AF 2C 32 43 20 AF B5 30 00 00 00 03 06 00 01
A2: 01 AF 2C 32 43 20 AF B5 30 00 00 00 03 06 00 02
C: 5F 13 1C E8 FF C1 D0 C6 9B 92 4E 47 B5 27 09 4C
   2B C8 5C F0 57 B9 1D E5 9B 33 4E 9A 31 81 D2 46
S0: C8 E7 95 98 8E 7B 1A 69 48 2A 3B EF CE 43 C5 A7
CipherText: 27 45 28 FA FE C1 D1 E3 9B FE 9E 1C B2 27 09 70
            24 C9 5C 41 53 27 1A C9 9B
U: 83 5A 26 AF 3D CB 93 5D
FinaData encrypted Payload: 27 45 28 FA FE C1 D1 E3 9B FE 9E 1C B2 27 09 70
                           24 C9 5C 41 53 27 1A C9 9B 83 5A 26 AF 3D CB 93 5D
```

Third Ranging block:

```
STS_Index for Pre Poll: 07 5B D1 B3
Frame Counter for Pre Poll: 00 00 00 04
CMAC-Data(URSK_KT): 00 00 00 01 55 52 53 4B 5F 4B 54 00 00 00 00 00
                   00 01 01 01 00 01 00 01 01 00 01 01 01 01 00 01
                   00 00 00 01 01 00 01 01 00 00 01 01 00 00 01 00
CMAC-Data(URSK_KT): 00 00 00 02 55 52 53 4B 5F 4B 54 00 00 00 00 00
                   00 01 01 01 00 01 00 01 01 00 01 01 01 01 00 01
                   00 00 00 01 01 00 01 01 00 00 01 01 00 00 01 00
URSK_KT: 2F B7 E8 39 2D E1 00 6E 6C CB 83 83 AB 12 B5 C0
        29 85 F4 87 65 F1 C0 B7 89 7F 61 1C B1 67 D8 0D
```

```
1      CMAC-Data(URSK): 00 00 00 01 55 52 53 4B 00 79 E8 65 18 6C BD 86
2      4B 95 56 82 C5 1C DD 34 F8 00 00 00 00 00 00 80
3      dURSK: A2 B7 89 A9 27 0E 74 85 D5 D9 48 D1 73 1B 92 4B
4      CMAC-Data(UDSK): 00 00 00 01 55 44 53 4B 00 79 E8 65 18 6C BD 86
5      4B 95 56 82 C5 1C DD 34 F8 00 00 00 00 00 00 80
6      dUDSK: CF 27 C7 7E 15 B9 E6 B9 88 62 90 32 FF 9D EE C9
7      PrePoll MHR: 49 2B D9 09 16 04 00 00 00 77 8B 07 9D AA 05 00
8      69 DF 04 01 0D 80 3F
9      PrePoll Payload: 78 56 34 12 B4 D1 5B 07 02 00 01 25 00
10     AES-CCM* key K: 94 CE 7A 78 B6 43 D0 CA C4 E2 E4 F8 B5 F5 5A 4D
11     Nonce N: AF 2C 32 43 20 AF B5 30 00 00 00 04 06
12     AddAuthData: 00 17 49 2B D9 09 16 04 00 00 00 77 8B 07 9D AA
13     05 00 69 DF 04 01 0D 80 3F 00 00 00 00 00 00 00
14     PlainTextData: 78 56 34 12 B4 D1 5B 07 02 00 01 25 00 00 00 00
15     AuthData: 00 17 49 2B D9 09 16 04 00 00 00 77 8B 07 9D AA
16     05 00 69 DF 04 01 0D 80 3F 00 00 00 00 00 00 00
17     78 56 34 12 B4 D1 5B 07 02 00 01 25 00 00 00 00
18     B0: 59 AF 2C 32 43 20 AF B5 30 00 00 00 04 06 00 0D
19     X: 77 6D 21 76 CF AA D5 17 95 E7 C1 A0 D0 09 48 18
20     D5 5F B6 8F 77 C3 87 E6 31 CD A1 0C 49 C9 97 87
21     05 62 09 AD 05 80 F9 5B 25 76 7A A8 91 01 53 69
22     88 8E 6E 6E 50 AD FA 4D 36 31 F1 A6 7B A4 7C CA
23     T: 88 8E 6E 6E 50 AD FA 4D
24     A0: 01 AF 2C 32 43 20 AF B5 30 00 00 00 04 06 00 00
25     A1: 01 AF 2C 32 43 20 AF B5 30 00 00 00 04 06 00 01
26     C: AA 3A 5F C4 60 B9 7D 9D 43 60 01 22 99 E7 DF 11
27     S0: 49 68 F9 C5 37 C4 FD C6 D3 50 6F 48 65 D7 72 37
28     CipherText: D2 6C 6B D6 D4 68 26 9A 41 60 00 07 99
29     U: C1 E6 97 AB 67 69 07 8B
30     PrePoll encrypted Payload: D2 6C 6B D6 D4 68 26 9A 41 60 00 07 99 C1 E6 97
31     AB 67 69 07 8B
32     STS-V for      POLL PPDU: 79 E8 65 18 6C BD 86 4B 9C B2 54 79 1C DD 34 F8 -
33     1C DD 35 17
34     STS for      POLL PPDU: D9 9B 10 0A C2 38 BB 61 3C 9E 26 94 57 9D C9 C6 ..
35     CF C4 28 0D E1 09 5E 22 80 54 64 38 B0 75 CC 30
36     STS-V for 1. RESPONSE PPDU: 79 E8 65 18 6C BD 86 4B 9C B2 54 7A 1C DD 34 F8 -
37     1C DD 35 17
38     STS for 1. RESPONSE PPDU: DD 39 EC 72 E6 CD AC FF E1 17 3B AA 3E 77 3C B3 ..
39     4C F1 78 CB 1C 47 68 E8 9C 74 17 A0 25 5B 17 3D
```

```
1      STS-V for          FINAL PPDU: 79 E8 65 18 6C BD 86 4B 9C B2 54 7B 1C DD 34 F8 -
2                                1C DD 35 17
3      STS for          FINAL PPDU: 2F FD 4E F7 3A 8B EC 9E 6D 62 FF F8 B1 34 A0 7A ..
4                                53 D8 A3 FB 00 23 05 D7 AE 7D 5A DE 87 EB 87 C4
5      Round Index of next Ranging Block (Default Hopping): 00 0C
6      FinaData MHR: 49 2B D9 09 16 05 00 00 00 77 8B 07 9D AA 05 00
7                                69 DF 04 02 19 80 3F
8      FinaData Payload: 78 56 34 12 02 00 01 0C 00 B6 D1 5B 07 00 00 3C
9                                0F 01 00 C0 04 9E 07 2C 00
10     AES-CCM* key K: CF 27 C7 7E 15 B9 E6 B9 88 62 90 32 FF 9D EE C9
11     Nonce N: AF 2C 32 43 20 AF B5 30 00 00 00 05 06
12     AddAuthData: 00 17 49 2B D9 09 16 05 00 00 00 77 8B 07 9D AA
13                                05 00 69 DF 04 02 19 80 3F 00 00 00 00 00 00
14     PlainTextData: 78 56 34 12 02 00 01 0C 00 B6 D1 5B 07 00 00 3C
15                                0F 01 00 C0 04 9E 07 2C 00 00 00 00 00 00 00
16     AuthData: 00 17 49 2B D9 09 16 05 00 00 00 77 8B 07 9D AA
17                                05 00 69 DF 04 02 19 80 3F 00 00 00 00 00 00
18                                78 56 34 12 02 00 01 0C 00 B6 D1 5B 07 00 00 3C
19                                0F 01 00 C0 04 9E 07 2C 00 00 00 00 00 00 00
20     B0: 59 AF 2C 32 43 20 AF B5 30 00 00 00 05 06 00 19
21     X: 70 B8 B0 47 F2 D5 2F CF 2C CB E6 9E 75 08 FE FE
22         51 1C 21 74 E3 D8 D9 70 39 5D F6 6B 9E AB 49 7A
23         B7 49 31 26 50 97 E6 06 5E 77 67 5F 33 9E 52 3F
24         23 00 69 8F 0D EA 5A D7 82 72 86 09 D3 36 6A 96
25         46 09 CE AE 51 E6 AF 42 7A 3D B2 C0 AD 68 E6 B1
26     T: 46 09 CE AE 51 E6 AF 42
27     A0: 01 AF 2C 32 43 20 AF B5 30 00 00 00 05 06 00 00
28     A1: 01 AF 2C 32 43 20 AF B5 30 00 00 00 05 06 00 01
29     A2: 01 AF 2C 32 43 20 AF B5 30 00 00 00 05 06 00 02
30     C: 1E 33 E7 49 01 D8 64 E1 DB 6C 6B E0 6C 3B CF 1D
31         41 77 C7 8E 21 A1 52 FE D5 67 DA 56 3A 1B B9 C4
32     S0: E5 07 B5 67 1A A8 DB 47 51 92 17 AB 6B D7 AC 21
33     CipherText: 66 65 D3 5B 03 D8 65 ED DB DA BA BB 6B 3B CF 21
34                                4E 76 C7 4E 25 3F 55 D2 D5
35     U: A3 0E 7B C9 4B 4E 74 05
36     FinaData encrypted Payload: 66 65 D3 5B 03 D8 65 ED DB DA BA BB 6B 3B CF 21
37                                4E 76 C7 4E 25 3F 55 D2 D5 A3 0E 7B C9 4B 4E 74 05
38
```